SQL and Relational Databases, Prompt Engineering

Forge of LLM Empires

Luca Corrocher

Table of Contents

List of Figures

1. Introduction

ChainForge

ChainForge is an open-source dataflow prompt-engineering environment to test a model's robustness to prompt injection attacks. As LLMs are too big to comprehensively evaluate across all possible use cases [1] [2], it provides a solution to this dilemma in the MLOps field, functioning as a middle ground between the exploratory/discovery phase of LLMs and evaluation/testing with an intuitive and powerful interface.

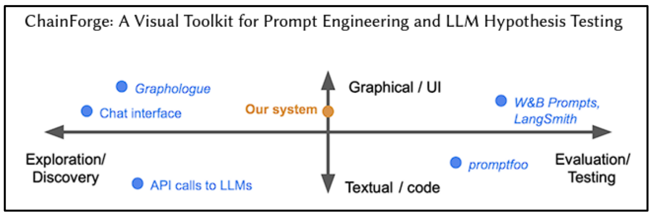
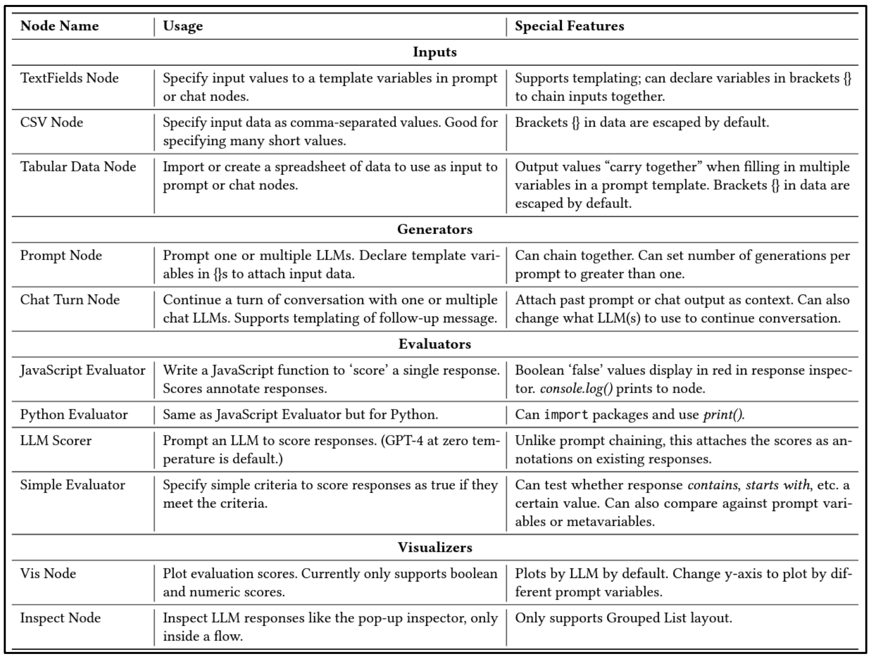


Figure 1. The emerging space of tools for LLM operations [3].

The tool uses a directed node graph-like topology to represent a logical step-by-step process of how multiple models respond to various circumstances, called chains. The output node of each chain can be analysed to give the user a calculated impression of each model accuracy.

The visual workflow tool incorporates a collection of nodes for workflow customisability and scalability; these include Input Data, Prompters, Evaluators Visualizers and Processors. This vast myriad of options allows users to test their workflow for any individual requirements - injecting a prompt nodes response as another input into another prompt node, multiple prompt node responses, back propagation of a response into an item node, the combinations are endless [4].



Node Name	Usage	Special Features
	Inputs	
TextFields Node	Specify input values to a template variables in prompt or chat nodes.	Supports templating; can declare variables in brackets {} to chain inputs together.
CSV Node	Specify input data as comma-separated values. Good for specifying many short values.	Brackets {} in data are escaped by default.
Tabular Data Node	Import or create a spreadsheet of data to use as input to prompt or chat nodes.	Output values "carry together" when filling in multiple variables in a prompt template. Brackets {} in data are escaped by default.
	Generators	
Prompt Node	Prompt one or multiple LLMs. Declare template variables in {}s to attach input data.	Can chain together. Can set number of generations per prompt to greater than one.
Chat Turn Node	Continue a turn of conversation with one or multiple chat LLMs. Supports templating of follow-up message.	Attach past prompt or chat output as context. Can also change what LLM(s) to use to continue conversation.
	Evaluators	
JavaScript Evaluator	Write a JavaScript function to 'score' a single response. Scores annotate responses.	Boolean 'false' values display in red in response inspector. *console.log()* prints to node.
Python Evaluator	Same as JavaScript Evaluator but for Python.	Can `import` packages and use *print()*.
LLM Scorer	Prompt an LLM to score responses. (GPT-4 at zero temperature is default.)	Unlike prompt chaining, this attaches the scores as annotations on existing responses.
Simple Evaluator	Specify simple criteria to score responses as true if they meet the criteria.	Can test whether response *contains*, *starts with*, etc. a certain value. Can also compare against prompt variables or metavariables.
	Visualizers	
Vis Node	Plot evaluation scores. Currently only supports boolean and numeric scores.	Plots by LLM by default. Change y-axis to plot by different prompt variables.
Inspect Node	Inspect LLM responses like the pop-up inspector, only inside a flow.	Only supports Grouped List layout.

Figure 2. The nodes in ChainForge, grouped by type [3]

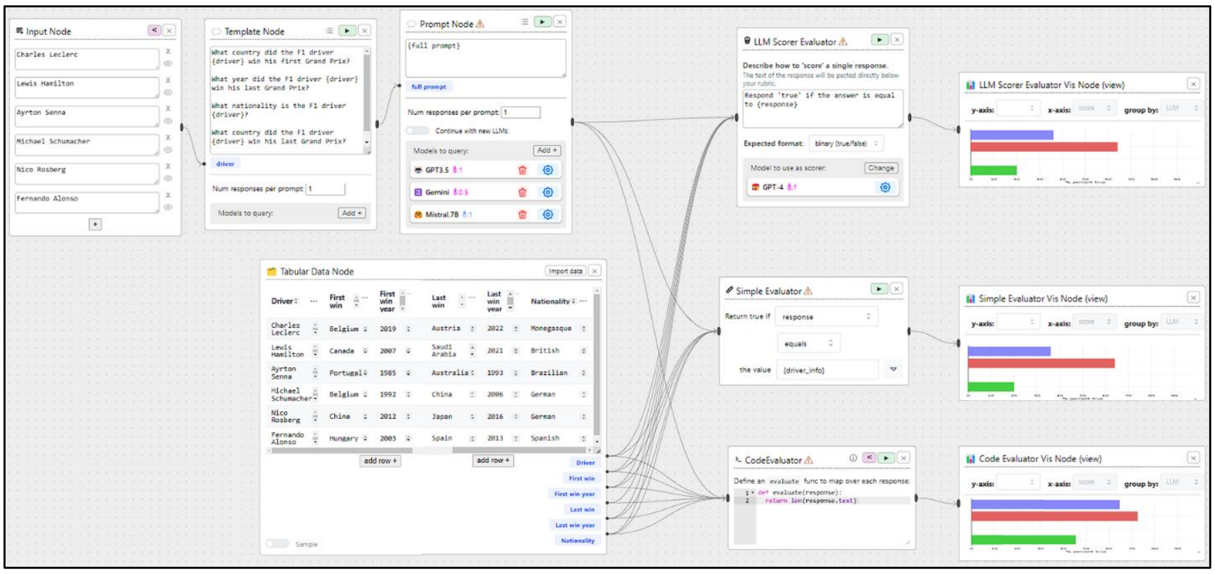
Project Vision

Although ChainForge is built upon a document-based NoSQL architecture [5] and upon consideration of prospective future of databases such as NewSQL [6], I propose a different approach using a relational database management system (RDBMS), which has additional features.

The design accommodates JSON formats of data stored as a Tabular Data Node and the Input Node contain attribute values of the Tabular Data Node. These work in combination by generating a full prompt from a prompt template, an Input Node value, and a Tabular Data Node attribute. This full prompt can then be queried multiple times by multiple large language models to generate multiple responses. Each response will be evaluated differently based on the parameters of the specific Evaluator Node. The results for each node can be grouped by the model per response to obtain a model performance accuracy metric for each model.

Additionally, the LLM Scorer Evaluator can hold more than one evaluator model, with temperature variation if desired (however temperature of 0 is preferred for less entropy). The LLM evaluator responses may vary between the evaluator models, and variability between datasets. A model tested on a diverse dataset is more likely to be more robust and generalize better to unseen prompts.

This design still allows for Prompt Node injection, but it has the ability to autogenerate multiple full prompts from template prompts depending on the number of Tabular Data Node attributes in a workflow. The incentive behind this is that a larger dataset of full prompts will result in a larger dataset of prompt responses by each model, which in turn corresponds to a more accurate model performance [7]. The variability of responses reduces bias, avoids overfitting, and may identify complex patterns and edge cases not previously considered.



Input Node:

Charles Leclerc

Lewis Hamilton

Ayrton Senna

Michael Schumacher

Nico Rosberg

Fernando Alonso

Template Node:

```
What country did the F1 driver {driver} win his first Grand Prix?

What year did the F1 driver {driver} win his last Grand Prix?

What nationality is the F1 driver {driver}?

What country did the F1 driver {driver} win his last Grand Prix?
```

Num responses per prompt: 1

Prompt Node:

```
{full prompt}
```

Num responses per prompt: 1

Tabular Data Node:

Driver	First win	First win year	Last win	Last win year	Nationality
Charles Leclerc	Belgium	2019	Austria	2022	Monegasque
Lewis Hamilton	Canada	2007	Saudi Arabia	2021	British
Ayrton Senna	Portugal	1985	Australia	1993	Brazilian
Michael Schumacher	Belgium	1992	China	2006	German
Nico Rosberg	China	2012	Japan	2016	German
Fernando Alonso	Hungary	2003	Spain	2013	Spanish

LLM Scorer Evaluator:

Describe how to 'score' a single response.
The text of the response will be pasted directly below your rubric.

```
Respond 'true' if the answer is equal to {response}
```

Expected format: binary (true/false)

Model to use as scorer: GPT-4

Simple Evaluator:

Return true if response equals the value {driver_info}

CodeEvaluator:

Define an evaluate func to map over each response:

```python
def evaluate(response):
  return len(response.text)
```

Figure 3. Evaluation logic of workflow using from template prompt for LLM injection.

Node convergence and divergence is a feature which allows for parallel processing of data in layers, feature combination. As seen in Figure 1, the Input Node and Tabular Data Node values converge to generate a full prompt and the prompt responses records diverge to each Evaluator Node. The Vis Nodes have the ability to verify the multiple responses generated per model and converge the Boolean result into a visual representation for comparison.

In summary, this RDBMS design allows for variability in Prompt Nodes through auto generation of full prompts from templates, divergence of responses returned by each model based on the prompt. The outcome results in an efficient and powerful platform to diagnose model flaws, stress test and analyse the limitations of each model. I am confident of its design which I will review in the following sections.

2. Database Plan: A Schematic View

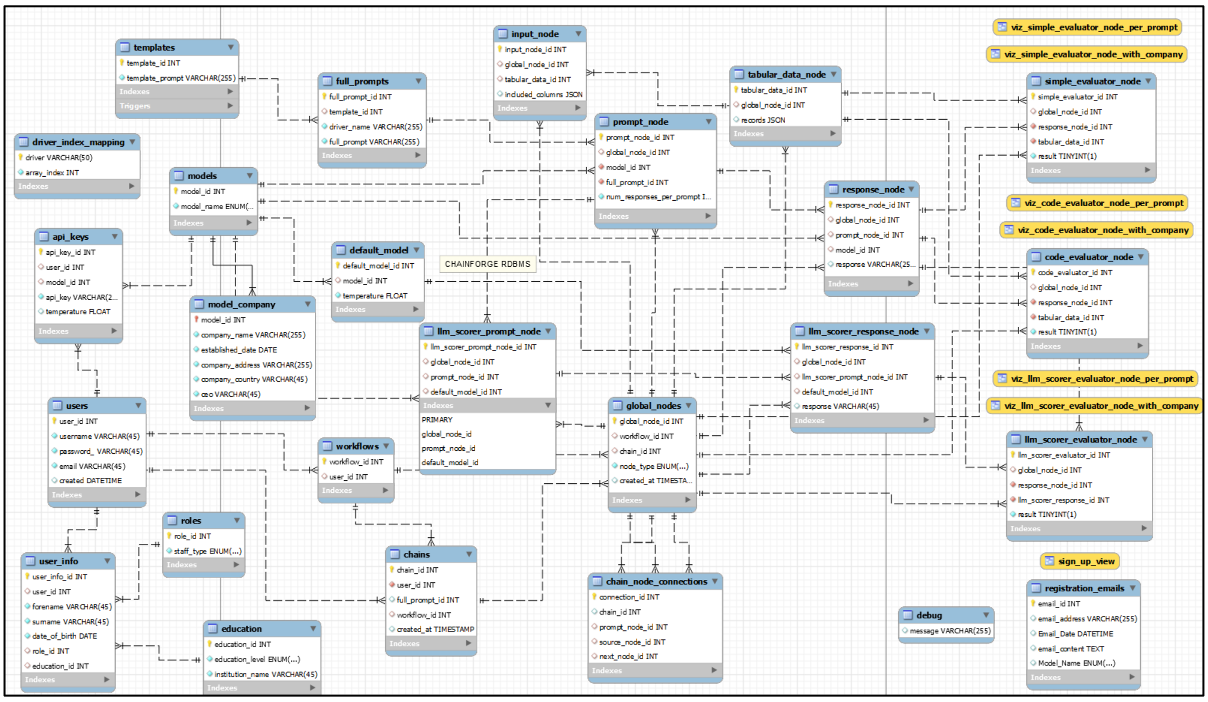


Figure 4. ER diagram of proposed ChainForge database design.

Figure 4. is the proposed ER diagram of the ChainForge RDBMS. Only in early development, it contains 26 tables, 7 views, and 8 procedural elements. Its design can account for the chain topology representation of Figure 3. in the ChainForge interface but can easily be suited to an individual chain topology.

The central tables in this design are the users, models, and the specific nodes tables which function as the children of the global_nodes table, generalising the characteristics of the nodes. The chain_node_connections table stores the related data to create the chain topology. This topology can be queried from a predefined query that has been created (L 1489 in .sql file).

The Viz Nodes in the ChainForge platform have been adapted not as tables, but as views in this RDBMS. They show a dynamic representation of the records in the Evaluator Nodes.

This RDBMS is highly capable and a fully functional database – the only requirement the client has to do is add a JSON object as a record in the Tabular Data Node and choose how to generate the template prompt; the remaining steps which lead to the results dataset and visualisation is taken care of by implementing each specific model API to generate the responses and later analyse the model performance.

The corresponding .sql and mwb. file have been shared with you and your team and have been created in MySQL with MySQL Workbench [8]. The .sql source file has been appropriately commented, should you wish to understand my thought process for each table and function.

3. Database Structure: A Normalized View

This section describes the design of the core set of tables which are pivotal to the ChainForge RDBMS functionality. Each table is interrelated to sub tables which condenses the core information for each process. The main components of the RDBMS design consist of the following generic tables (some are broken down further):

users: the account for the chain and workflow.

models: information on LLM models chosen.

prompt nodes: contains a full prompt for prompt injection.

response nodes: contains records of simulated LLM model responses relating to a specific prompt node record.

evaluator nodes: compares the LLM responses with the correct responses, either queried from the database or from an accurate default LLM model.

viz nodes (views): visualisations of the records of the evaluator node records.

chain node connections: forms a chain in a workflow based on a central nodes table, global nodes, which relates to each child node. This forms the converging/diverging graph topology that is seen in the ChainForge interface (Figure 3.).

Each of these tables are normalised in Boyce Codd Normal Form (BCNF) [9] which minimised data redundancy, improves query performance and it easily maintainable and consistent for a large scale user base [10] [11].

Users Table

This contains the interrelated records of each user's account details, username, what date/time the user was enrolled in the system. A user can choose many models, therefore many API keys. The user_id is the primary super key (PK) which relates to its surrounding tables. It relates to the following tables:

user_info: one to one (1 – 1). user_info_id [PK], education_id, role_id [FK]. Describes the additional information about the user and attempts to categorise the users with the auxiliary tables (roles and education)

api_keys: one to many (1 – m), api_key_id [PK], model_id [FK]. Describes the api key of the model that the user selects in the workflow. One user can have many api keys depending on the number of models they incorporate.

workflows: 1 – m, workflow_id [PK], user_id [FK]. One user can have many workflows.

chains: 1 – m, chain_id [PK]. One user can have many chains in each workflow.

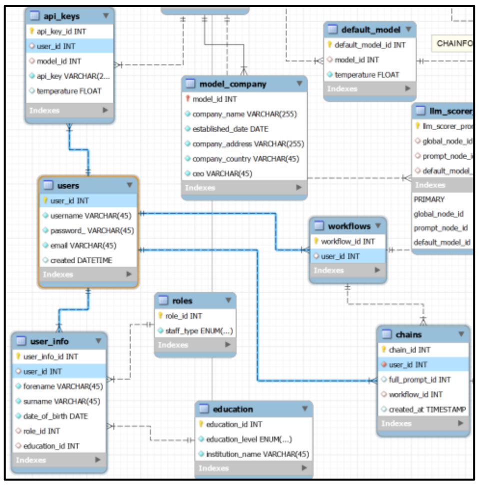
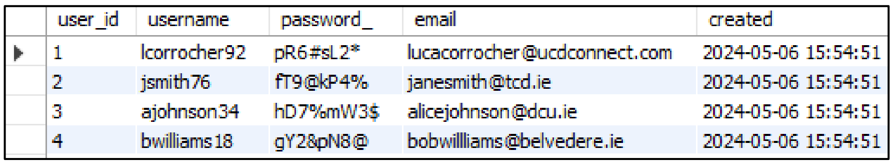


Figure 5. ER diagram – users relationships.



	user_id	username	password_	email	created
▶	1	lcorrocher92	pR6#sL2*	lucacorrocher@ucdconnect.com	2024-05-06 15:54:51
	2	jsmith76	fT9@kP4%	janesmith@tcd.ie	2024-05-06 15:54:51
	3	ajohnson34	hD7%mW3$	alicejohnson@dcu.ie	2024-05-06 15:54:51
	4	bwilliams18	gY2&pN8@	bobwillliams@belvedere.ie	2024-05-06 15:54:51

Figure 6. Sample records from users.

Models Table

In ChainForge, the user can choose several models for evaluation testing. Each model has a unique model id. An auxiliary table segregates the default model from the testing models, with the default set at temperature 0. This contains the interrelated records of each model_id and the corresponding model type e.g. GPT3.5 Turbo. The model_id is the primary super key (PK) which relates to its surrounding tables. It relates to the following tables:

api_keys: 1 – m, api_key_id [PK]. One model can have many api keys for many users.

default_model: 1 – m, default_model_id [PK]. One model can have many default models at default temperature 0.

model_company: m – 1, model_id [PK]. Many models can have one company e.g. Chat GPT 3.5 Turbo and GPT4 belong to OpenAI.

prompt_node: 1 – m, prompt_node_id [PK]. One model can relate to many prompt node records. A minimum of one model is needed for comparison.

llm_scorer_prompt_node: 1 – m, llm_scorer_prompt_node_id [PK]. One model can relate to many llm scorer prompt node records.

response_node: 1 – m, response_node_id [PK]. One model can relate to many response node records.

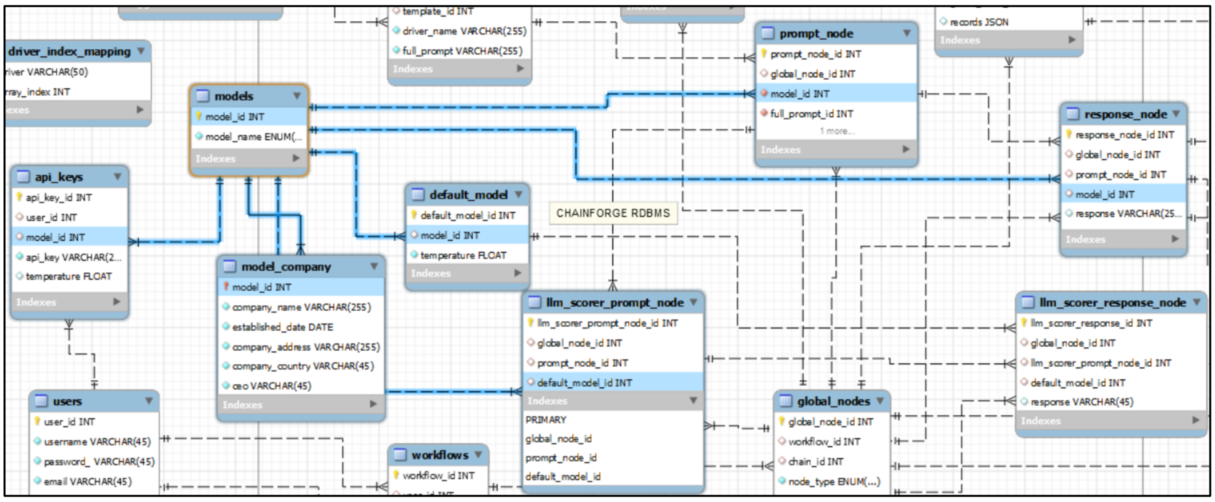
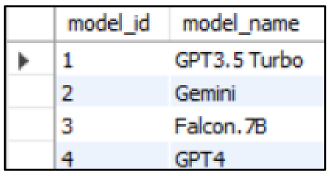


Figure 7. ER diagram - models relationships.



	model_id	model_name
▶	1	GPT3.5 Turbo
	2	Gemini
	3	Falcon.7B
	4	GPT4

Figure 8. Current model records in database.

Prompt Node Table

The prompt node incorporates the models to query, the full prompt question and the number of responses per prompt all in one table. It contains the interrelated records of each prompt node. The super key is the prompt_node_id and it relates to the following surrounding tables:

models: m – 1, models_id [PK]. Many prompt node records will query one type of model.

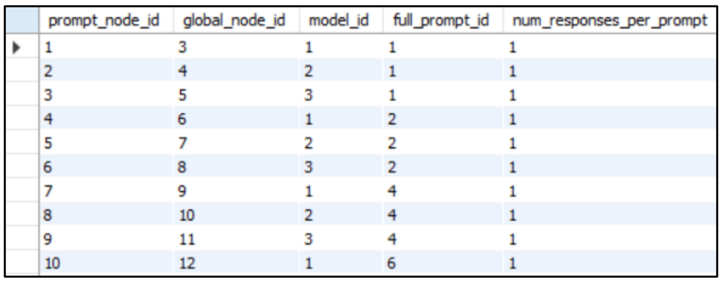
full_prompts: m – 1, full_prompt_id [PK]. Many prompt nodes are based on 1 full prompt record. Each full prompt record is an implementation of a template prompt.

llm_scorer_prompt_node: 1 – m, llm_scorer_prompt_node_id [PK]. One prompt node record can have many llm_scorer_prompt_node records, depending on the number of default models they have (1 in our case).

response_node: 1 – m, response_node_id [PK]. One prompt node record can have multiple response nodes records, depending on the number of responses per prompt and number of models queried.

global_nodes: m – 1, global_node_id [PK]. Many prompt nodes can relate to one global node type.

In the populated database, I have implemented 13 full prompts and 3 models to query. As each model only responds once per prompt the total available prompt node records is 13 x 3 = 39.



prompt_node_id	global_node_id	model_id	full_prompt_id	num_responses_per_prompt
1	3	1	1	1
2	4	2	1	1
3	5	3	1	1
4	6	1	2	1
5	7	2	2	1
6	8	3	2	1
7	9	1	4	1
8	10	2	4	1
9	11	3	4	1
10	12	1	6	1

Figure 9. Sample of prompt node records

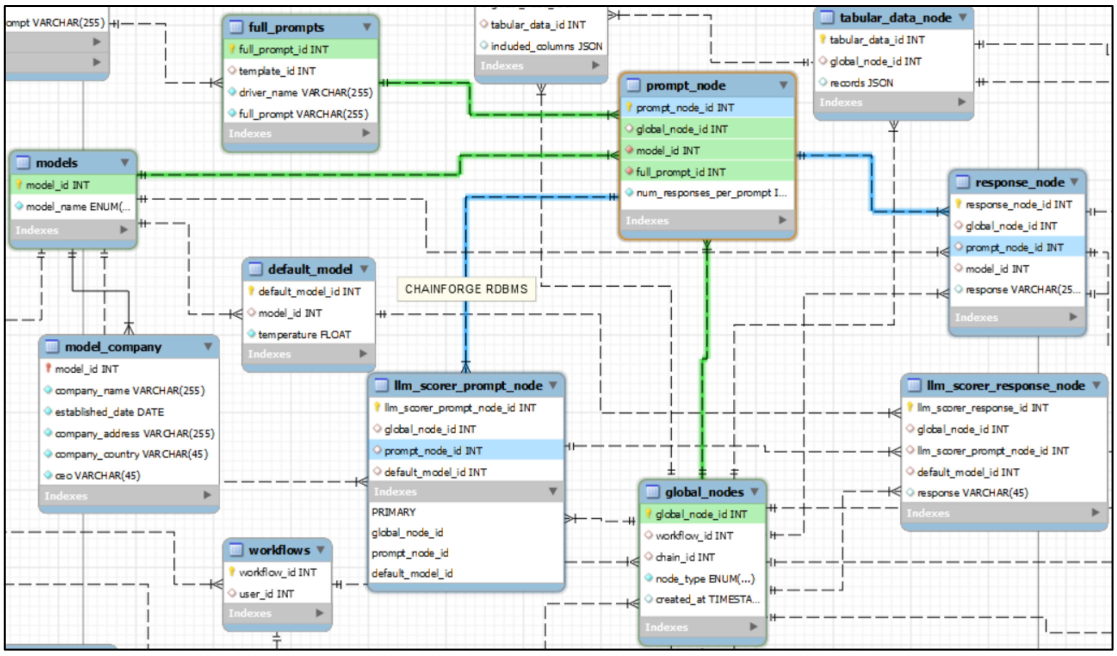


Figure 10. ER diagram - prompt node relationships.

Response Node Table

Once the prompt node has been executed and linked to the desired evaluator nodes in the ChainForge interface, each prompt node record queries an individual LLM to receive a response record. The response node table contains the interrelated records of each response node. It contains the super key response_node_id and relates to the following surrounding tables:

models: m – 1 model_id [PK]. Many response node records relate to only one model.

prompt_node: m – 1, prompt_node_id [PK]. Many response node records relate to only one prompt node.

global_nodes: m – 1, global_node_id [PK]. Many response node records relate to one type of global node.

simple_evaluator_node: 1 – m, simple_evaluator_id [PK]. Each response node record relates to many simple evaluator nodes (SEN). As each response record only has one model that it has queried the response from this model will be passed to the evaluator node.

code_evaluator_node: 1 - m, code_evaluator_id [PK]. Identical to SEN.

llm_scorer_evaluator_node: 1- m, llm_scorer_evaluator_id [PK]. Identical to SEN.

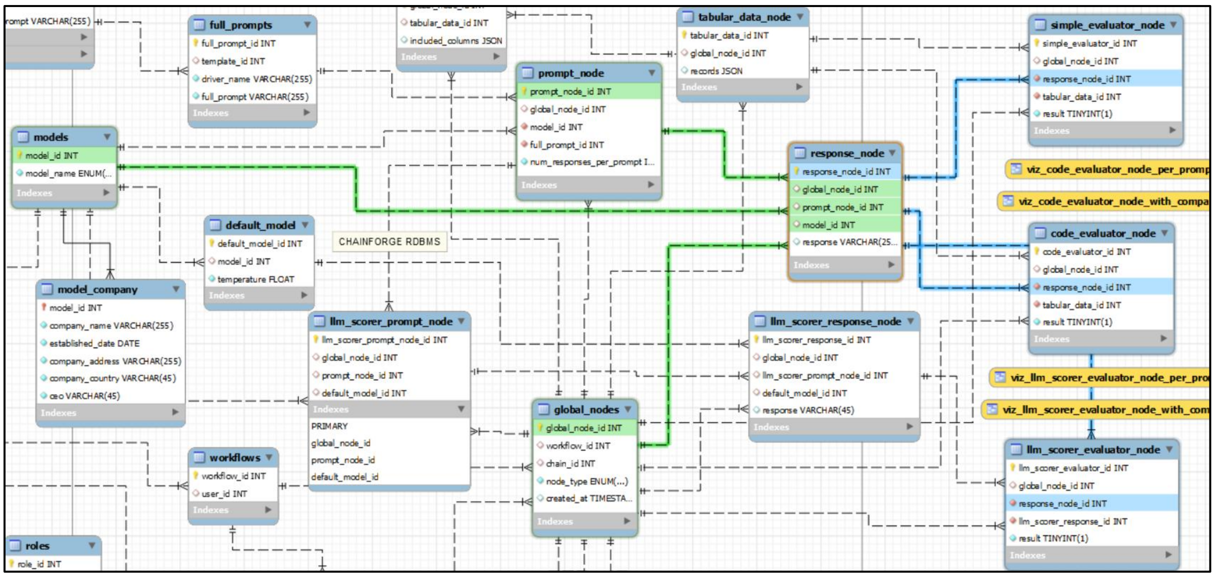
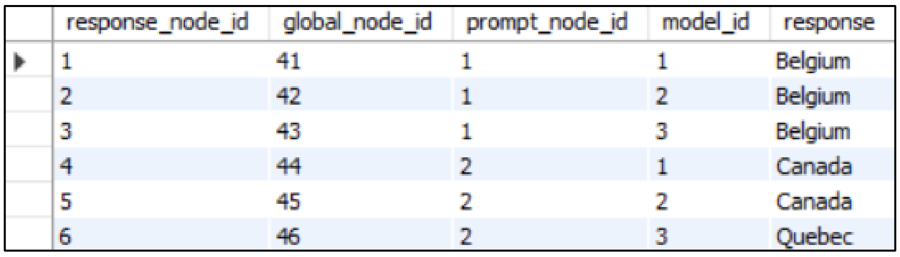


Figure 11. ER diagram - response_node table. Records contain the answers by each model for a full prompt.

My implementation consists of 39 response node records, one for each model, prompt, and number of responses. Each response will have to be Boolean evaluated in the next stage. For example, response node 1 relates to the prompt node id 1, which is the question *"What country did the F1 driver Charles Leclerc win his first grand prix?".* It queries the first model, which returns a varchar response "*Belgium*". Note the response for model id 3 in record 6 returns a different response from record 4 and 5. These responses will be evaluated in the evaluator node.



	response_node_id	global_node_id	prompt_node_id	model_id	response
▶	1	41	1	1	Belgium
	2	42	1	2	Belgium
	3	43	1	3	Belgium
	4	44	2	1	Canada
	5	45	2	2	Canada
	6	46	2	3	Quebec

Figure 12. Sample of response node records.

Simple/Code/LLM Scorer Node Tables

These are very similar tables which have a different approach to verifying each response. The simple evaluator node compares each response to the tabular data node attribute value and checks if the two values are equal. The code evaluator also compares the response with the tabular data node value but evaluated it as correct if the response length is equal to the value. This will lead to a higher false positive rate and a decreased precision rate and is a less reliable classification system. Finally, the LLM scorer node also uses a Boolean classification to evaluate each response, however it compares the response to a value that is the output of the accurate default LLM model.

The simple evaluator and LLM scorer evaluator tend to converge on the same correct classification, however, the tabular data in never changes and always maintains a zero-entropy characteristic, whereas the LLM response node may return a false positive on some occasions.

Each evaluator node contains the super key of their respective node id and relates to the global nodes table as well as either the tabular data node with the correct responses to the prompt, or the LLM scorer response node.

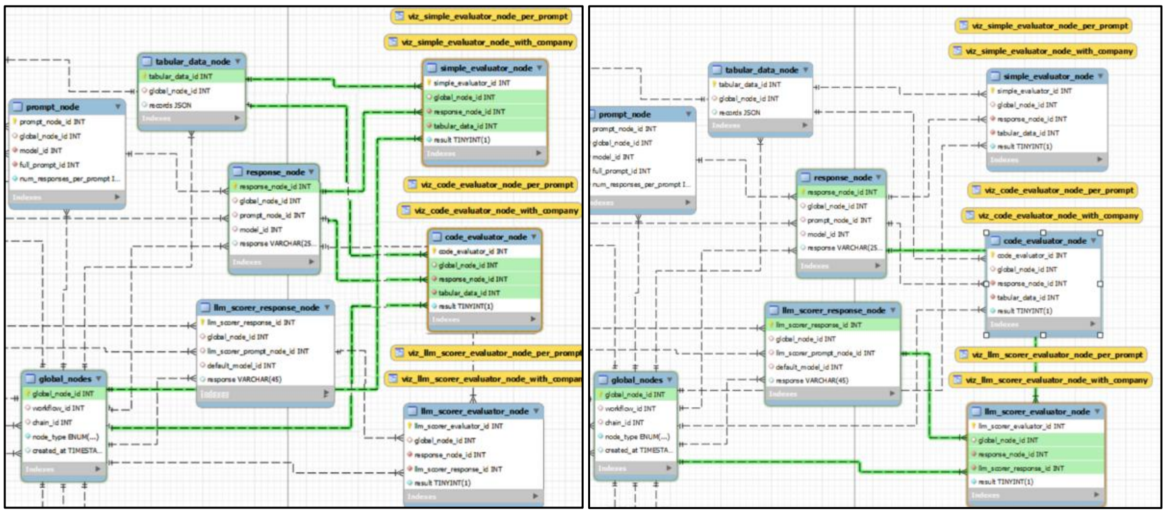
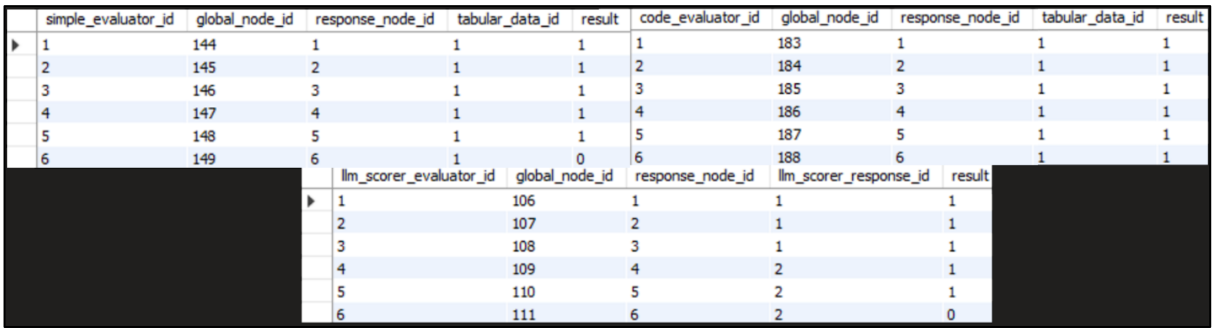

viz_simple_evaluator_node_per_prompt
viz_simple_evaluator_node_with_company
tabular_data_node
simple_evaluator_node
prompt_node
response_node
viz_code_evaluator_node_per_prompt
viz_code_evaluator_node_with_company
code_evaluator_node
llm_scorer_response_node
viz_llm_scorer_evaluator_node_per_prompt
viz_llm_scorer_evaluator_node_with_company
global_nodes
llm_scorer_evaluator_node

Figure 13. Figure 13. ER diagram - simple evaluator node relationships. Code evaluator and LLM scorer evaluator nodes are similar but code has higher false positive, LLM can have very low false positive rate.

Displayed below are the first six records for the simple/code/LLM scorer evaluators. Notice how the simple evaluator and LLM scorer evaluator classifies record six as false, whereas the code evaluator marks classify it as true based on the equal word length. Code evaluator's result in record six is a false positive.



simple_evaluator_id	global_node_id	response_node_id	tabular_data_id	result
1	144	1	1	1
2	145	2	1	1
3	146	3	1	1
4	147	4	1	1
5	148	5	1	1
6	149	6	1	0

code_evaluator_id	global_node_id	response_node_id	tabular_data_id	result
1	183	1	1	1
2	184	2	1	1
3	185	3	1	1
4	186	4	1	1
5	187	5	1	1
6	188	6	1	1

llm_scorer_evaluator_id	global_node_id	response_node_id	llm_scorer_response_id	result
1	106	1	1	1
2	107	2	1	1
3	108	3	1	1
4	109	4	2	1
5	110	5	2	1
6	111	6	2	0

Figure 14. Comparison between results of simple/code/LLM scorer evaluator node records.

Global Nodes and Chain Node Connections

The global_nodes table acts as the central connecting table between all of the nodes, the workflow, the chains and the chain links. It relates to all of the child nodes via its primary super key global_node_id with a one-to-many relationship i.e. many child nodes link to the one global node table. Each workflow can contain many chains with a collection of child nodes. The chains can be represented using the global_node_id or the child node_id (we see this in the queries later). The chain node connections table forms an adjacency list graph, which represents a full chain.

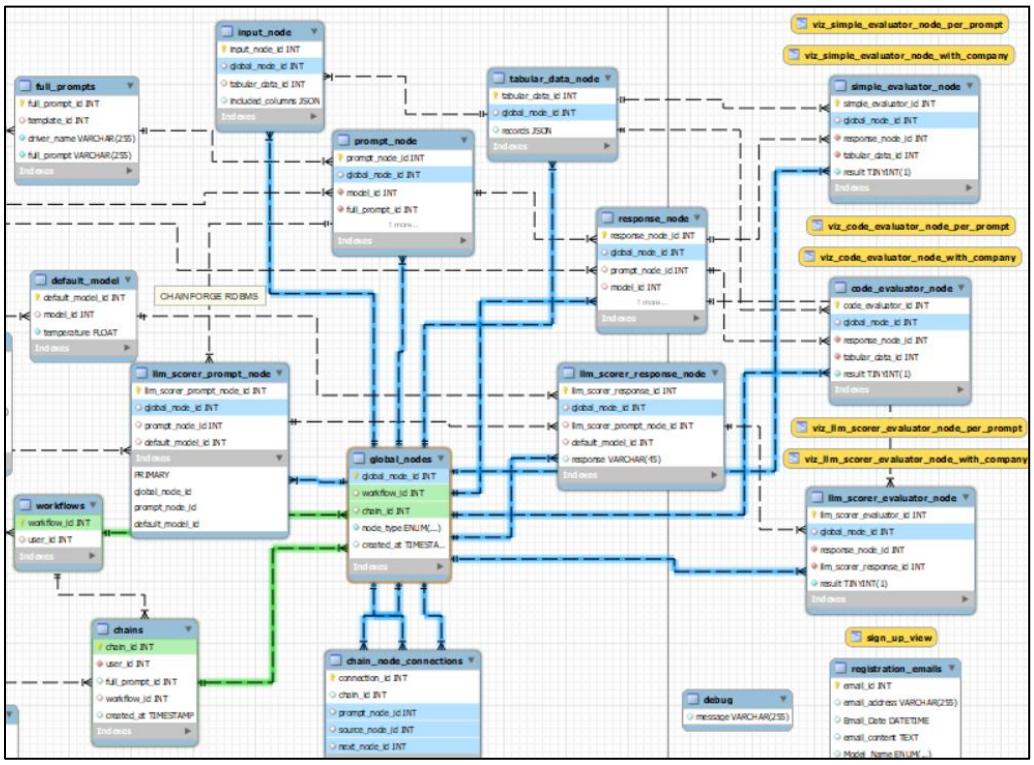
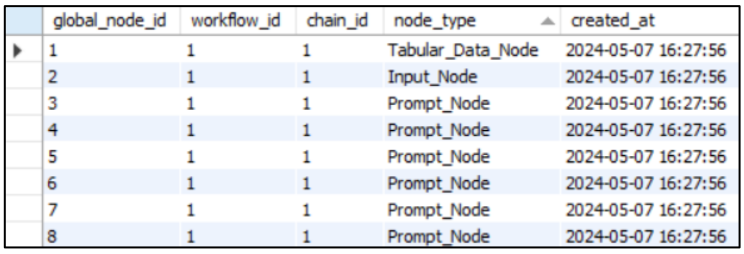


Figure 15. ER diagram – global nodes relationships.

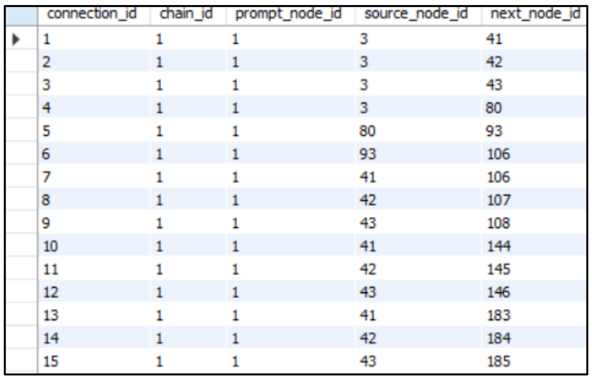
It was essential to have a global node identifier and a specific child node identifier for this RDBMS. This way queries can be easily grouped by each specific node table or the global table. There are 232 global nodes in this table in the current implementation.



global_node_id	workflow_id	chain_id	node_type	created_at
1	1	1	Tabular_Data_Node	2024-05-07 16:27:56
2	1	1	Input_Node	2024-05-07 16:27:56
3	1	1	Prompt_Node	2024-05-07 16:27:56
4	1	1	Prompt_Node	2024-05-07 16:27:56
5	1	1	Prompt_Node	2024-05-07 16:27:56
6	1	1	Prompt_Node	2024-05-07 16:27:56
7	1	1	Prompt_Node	2024-05-07 16:27:56
8	1	1	Prompt_Node	2024-05-07 16:27:56

Figure 16. Sample records from global nodes table.

Figure 17. below shows a sample set of records for chain node connections. Each In the implementation 48 chain node connections exist, 24 for chain_id 1 and 24 for chain_id 2. Each chain node connection record has been designed similar to a diverging linked list, with a global node identifier for the source node and the next node. The first four records below show source node 3 diverging to three next node objects. Other nodes may converge into one node object, three response nodes to one evaluator node, for example.



connection_id	chain_id	prompt_node_id	source_node_id	next_node_id
1	1	1	3	41
2	1	1	3	42
3	1	1	3	43
4	1	1	3	80
5	1	1	80	93
6	1	1	93	106
7	1	1	41	106
8	1	1	42	107
9	1	1	43	108
10	1	1	41	144
11	1	1	42	145
12	1	1	43	146
13	1	1	41	183
14	1	1	42	184
15	1	1	43	185

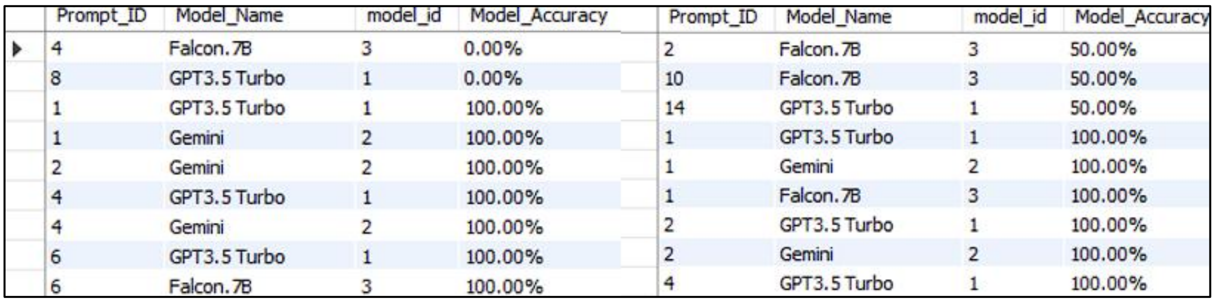
Figure 17. Sample records for chain node connections table.

4. Database Views

Views are logical queries presented as updating virtual tables to aim to provide the user with insightful information. The views incorporated into the project act as the Viz Node in ChainForge with dynamic information based on the evaluator node tables. The results of the Viz Nodes will never be queried, which made this ideal to represent as views. The views provided give the user an insight into the model performance accuracy in various representations.

Model Performance Accuracy Comparison Per Prompt

The first three views display the model performance accuracy according to the individual prompt. The below records show the model performance accuracy based on the model and the specific prompt. One can deduce that the simple/LLM evaluators fail in the comparison between for the response for prompts 4 and 5. The lowest values for the code evaluator was the classification in prompt 2 and 10 for the Falcon. 7B.



Prompt_ID	Model_Name	model_id	Model_Accuracy
4	Falcon.7B	3	0.00%
8	GPT3.5 Turbo	1	0.00%
1	GPT3.5 Turbo	1	100.00%
1	Gemini	2	100.00%
2	Gemini	2	100.00%
4	GPT3.5 Turbo	1	100.00%
4	Gemini	2	100.00%
6	GPT3.5 Turbo	1	100.00%
6	Falcon.7B	3	100.00%

Prompt_ID	Model_Name	model_id	Model_Accuracy
2	Falcon.7B	3	50.00%
10	Falcon.7B	3	50.00%
14	GPT3.5 Turbo	1	50.00%
1	GPT3.5 Turbo	1	100.00%
1	Gemini	2	100.00%
1	Falcon.7B	3	100.00%
2	GPT3.5 Turbo	1	100.00%
2	Gemini	2	100.00%
4	GPT3.5 Turbo	1	100.00%

Figure 18. Sample records from simple/LLM evaluator view (left) and code evaluator view (right)

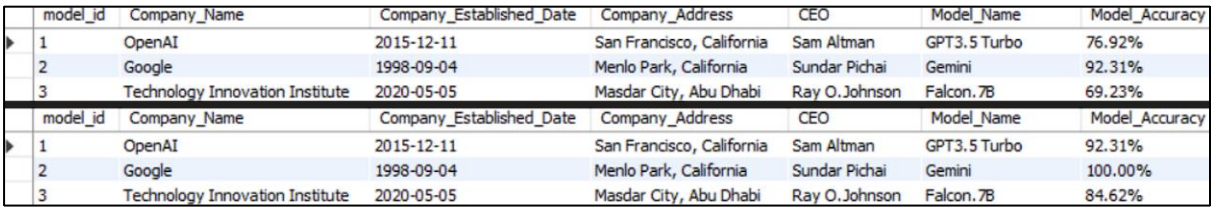
Model Performance Accuracy Comparison Per Model

The next view provides a useful insight into the average performance accuracy of each model. The models implemented in the workflow were OpenAI's GPT3.5 Turbo, Google's Gemini and TII's Falcon.7B. Additional company information is represented in each record. For demonstration of the database interaction, 12 prompts were injected into each model (3), so every correct evaluation corresponds to 8.6%.

The reason that the top view is identical for the simple code evaluator and the LLM scorer is because the LLM scorer contained no false positives in the prompts injected to the default LLM model. For a larger data set of prompt injections, the LLM scorer evaluator may return an incorrect response.

In the top view, the model with the highest performance accuracy is Gemini at 92.31% and the lowest Falcon. 7B at 69.23%, with 11 out of 12 and 8 out of 12 prompt answered correctly respectively. There may be a relationship between current model performance accuracy and the years of experience the company has in the LLM field. This was simplified as the company founding year.

Another significant observation is that the model accuracy for the bottom view i.e. code evaluator for the exact same prompts was higher for Gemini than in the simple evaluator or LLM evaluator. This is because its evaluation criterion is less precise, therefore losing its accuracy. So, while it still reports a 100% model accuracy, it contains one false positive prompt.



model_id	Company_Name	Company_Established_Date	Company_Address	CEO	Model_Name	Model_Accuracy
1	OpenAI	2015-12-11	San Francisco, California	Sam Altman	GPT3.5 Turbo	76.92%
2	Google	1998-09-04	Menlo Park, California	Sundar Pichai	Gemini	92.31%
3	Technology Innovation Institute	2020-05-05	Masdar City, Abu Dhabi	Ray O.Johnson	Falcon.7B	69.23%

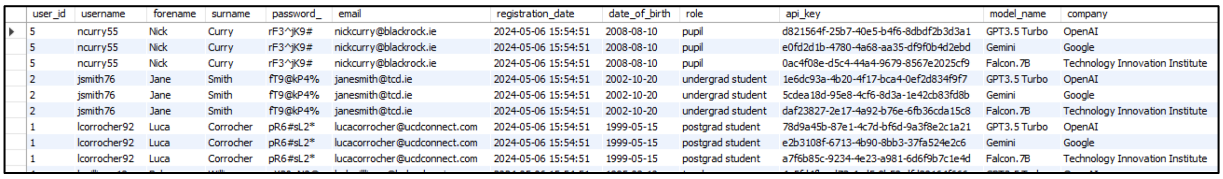
model_id	Company_Name	Company_Established_Date	Company_Address	CEO	Model_Name	Model_Accuracy
1	OpenAI	2015-12-11	San Francisco, California	Sam Altman	GPT3.5 Turbo	92.31%
2	Google	1998-09-04	Menlo Park, California	Sundar Pichai	Gemini	100.00%
3	Technology Innovation Institute	2020-05-05	Masdar City, Abu Dhabi	Ray O.Johnson	Falcon.7B	84.62%

Figure 19. Comparison of model performance accuracies for simple/LLM scorer evaluator (top) and code evaluator (bottom)

Sign-up View

Another view was created to collect the information when a user has signed up to a particular LLM. It gathers their username, personal details, email address, the sign-up registration date, the unique API key for the model and the role of the user. Any time a user fills in all this criterion, the view updates dynamically to represent this information. As each model and API key is different, as well as the potential model registration time are all unique, the records cannot be aggregated, but are simply unique.

As there are currently eight users in the RDBMS and they all use 3 separate modes, the number of records in the system equates to 8 x 3 = 24.



user_id	username	forename	surname	password_	email	registration_date	date_of_birth	role	api_key	model_name	company
5	ncurry55	Nick	Curry	rF3^jK9#	nickcurry@blackrock.ie	2024-05-06 15:54:51	2008-08-10	pupil	d821564f-25b7-40e5-b4f6-8dbdf2b3d3a1	GPT3.5 Turbo	OpenAI
5	ncurry55	Nick	Curry	rF3^jK9#	nickcurry@blackrock.ie	2024-05-06 15:54:51	2008-08-10	pupil	e0fd2d1b-4780-4a68-aa35-df9f0b4d2ebd	Gemini	Google
5	ncurry55	Nick	Curry	rF3^jK9#	nickcurry@blackrock.ie	2024-05-06 15:54:51	2008-08-10	pupil	0ac4f08e-d5c4-44a4-9679-8567e2025cf9	Falcon.7B	Technology Innovation Institute
2	jsmith76	Jane	Smith	fT9@kP4%	janesmith@tcd.ie	2024-05-06 15:54:51	2002-10-20	undergrad student	1e6dc93a-4b20-4f17-bca4-0ef2d834f9f7	GPT3.5 Turbo	OpenAI
2	jsmith76	Jane	Smith	fT9@kP4%	janesmith@tcd.ie	2024-05-06 15:54:51	2002-10-20	undergrad student	5cdea18d-95e8-4cf6-8d3a-1e42cb83fd8b	Gemini	Google
2	jsmith76	Jane	Smith	fT9@kP4%	janesmith@tcd.ie	2024-05-06 15:54:51	2002-10-20	undergrad student	daf23827-2e17-4a92-b76e-6fb36cda15c8	Falcon.7B	Technology Innovation Institute
1	lcorrocher92	Luca	Corrocher	pR6#sL2*	lucacorrocher@ucdconnect.com	2024-05-06 15:54:51	1999-05-15	postgrad student	78d9a45b-87e1-4c7d-bf6d-9a3f8e2c1a21	GPT3.5 Turbo	OpenAI
1	lcorrocher92	Luca	Corrocher	pR6#sL2*	lucacorrocher@ucdconnect.com	2024-05-06 15:54:51	1999-05-15	postgrad student	e2b3108f-6713-4b90-8bb3-37fa524e2c6	Gemini	Google
1	lcorrocher92	Luca	Corrocher	pR6#sL2*	lucacorrocher@ucdconnect.com	2024-05-06 15:54:51	1999-05-15	postgrad student	a7f6b85c-9234-4e23-a981-6d6f9b7c1e4d	Falcon.7B	Technology Innovation Institute

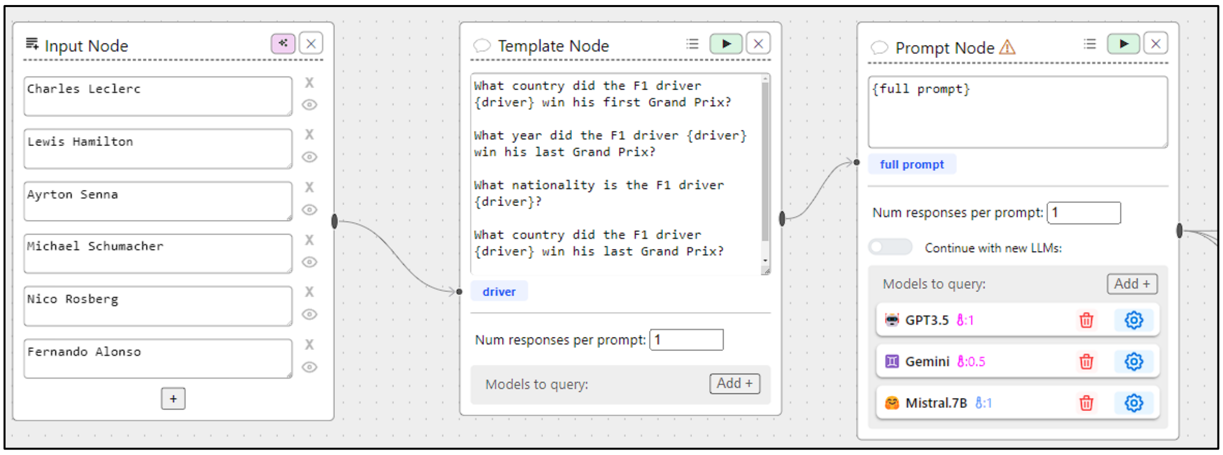
Figure 20. Sample records for the sign-up view.

5. Procedural Elements (PL/SQL)

Triggers

Generate Full Prompts

The generate full prompts is a trigger which automatically generates full prompts from templates by replacing the template key word with variables. In this implementation, the key word was denoted as *{driver}.* As six drivers currently exist in this tabular data node and the four template prompts have been created, the generate full prompts trigger iterates through each driver to create a full specific prompt, 24 in total. Whenever new entries are added to the templates table, six new full prompt entries will automatically be generated.



Input Node

```
Charles Leclerc
Lewis Hamilton
Ayrton Senna
Michael Schumacher
Nico Rosberg
Fernando Alonso
```

Template Node

```
What country did the F1 driver
{driver} win his first Grand Prix?

What year did the F1 driver {driver}
win his last Grand Prix?

What nationality is the F1 driver
{driver}?

What country did the F1 driver
{driver} win his last Grand Prix?
```

driver

Num responses per prompt: 1

Models to query:

Prompt Node

```
{full prompt}
```

full prompt

Num responses per prompt: 1

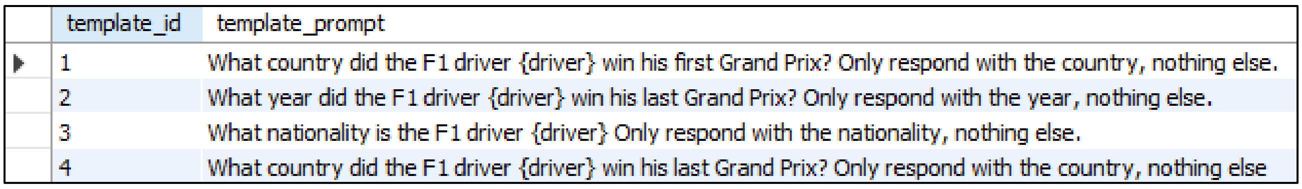
Continue with new LLMs:

Models to query:

- GPT3.5 1
- Gemini 0.5
- Mistral.7B 1

Figure 21. Visualisation of how trigger generates the full prompt from a template prompt.

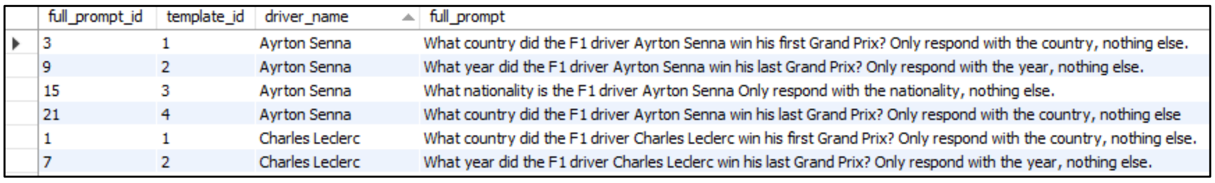
The template prompts that have been entered into the RDBMS are the following:



template_id	template_prompt
1	What country did the F1 driver {driver} win his first Grand Prix? Only respond with the country, nothing else.
2	What year did the F1 driver {driver} win his last Grand Prix? Only respond with the year, nothing else.
3	What nationality is the F1 driver {driver} Only respond with the nationality, nothing else.
4	What country did the F1 driver {driver} win his last Grand Prix? Only respond with the country, nothing else

Figure 22. All template prompt records in RDBMS.

These four templates generate the full prompts replacing the driver every time. A sample of these can be seen below.



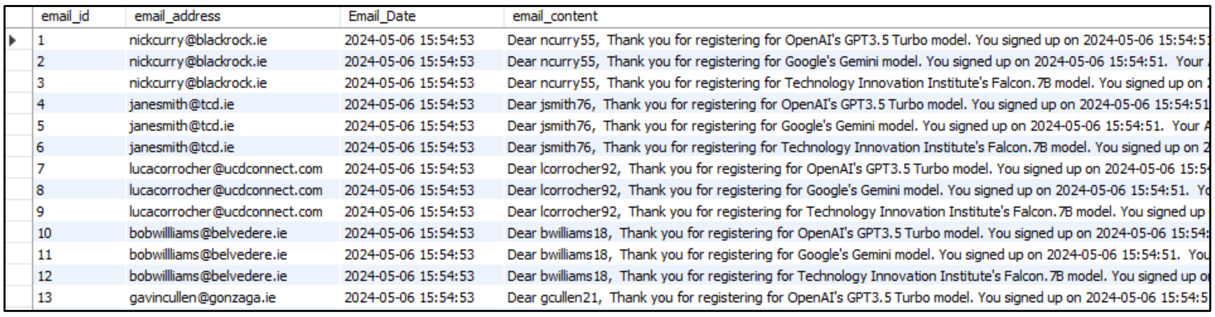
full_prompt_id	template_id	driver_name	full_prompt
3	1	Ayrton Senna	What country did the F1 driver Ayrton Senna win his first Grand Prix? Only respond with the country, nothing else.
9	2	Ayrton Senna	What year did the F1 driver Ayrton Senna win his last Grand Prix? Only respond with the year, nothing else.
15	3	Ayrton Senna	What nationality is the F1 driver Ayrton Senna Only respond with the nationality, nothing else.
21	4	Ayrton Senna	What country did the F1 driver Ayrton Senna win his last Grand Prix? Only respond with the country, nothing else
1	1	Charles Leclerc	What country did the F1 driver Charles Leclerc win his first Grand Prix? Only respond with the country, nothing else.
7	2	Charles Leclerc	What year did the F1 driver Charles Leclerc win his last Grand Prix? Only respond with the year, nothing else.

Figure 23. Sample of full prompts generated from introducing six drivers as variable.

Stored Procedures

Send Bulk Registration Emails

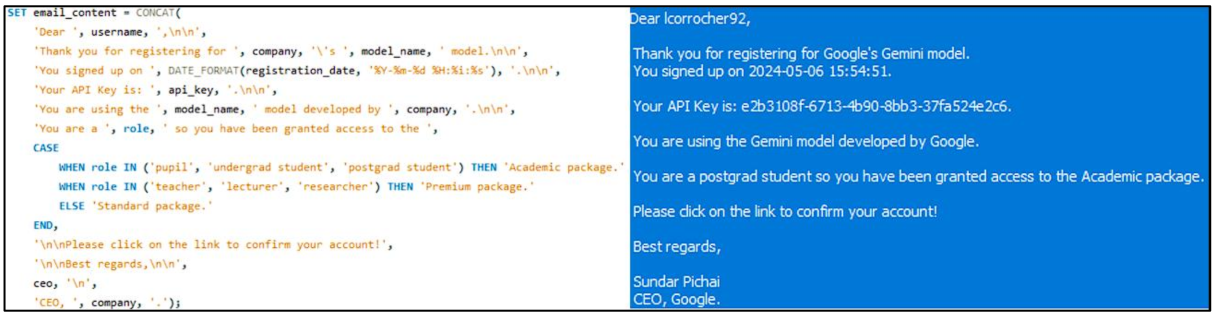
This stored procedure queries the database similar to the sign-up view to select the parameters to automatically generate an email for users that have signed up to the ChainForge RDBMS. It iterates through every record and inserts the records into the registration emails table. The generated email various depend on user, model chosen and education role and the registration package, company, CEO and model API key vary accordingly.



	email_id	email_address	Email_Date	email_content
▶	1	nickcurry@blackrock.ie	2024-05-06 15:54:53	Dear ncurry55, Thank you for registering for OpenAI's GPT3.5 Turbo model. You signed up on 2024-05-06 15:54:5
	2	nickcurry@blackrock.ie	2024-05-06 15:54:53	Dear ncurry55, Thank you for registering for Google's Gemini model. You signed up on 2024-05-06 15:54:51. Your
	3	nickcurry@blackrock.ie	2024-05-06 15:54:53	Dear ncurry55, Thank you for registering for Technology Innovation Institute's Falcon.7B model. You signed up on
	4	janesmith@tcd.ie	2024-05-06 15:54:53	Dear jsmith76, Thank you for registering for OpenAI's GPT3.5 Turbo model. You signed up on 2024-05-06 15:54:51
	5	janesmith@tcd.ie	2024-05-06 15:54:53	Dear jsmith76, Thank you for registering for Google's Gemini model. You signed up on 2024-05-06 15:54:51. Your
	6	janesmith@tcd.ie	2024-05-06 15:54:53	Dear jsmith76, Thank you for registering for Technology Innovation Institute's Falcon.7B model. You signed up on
	7	lucacorrocher@ucdconnect.com	2024-05-06 15:54:53	Dear lcorrocher92, Thank you for registering for OpenAI's GPT3.5 Turbo model. You signed up on 2024-05-06 15:5
	8	lucacorrocher@ucdconnect.com	2024-05-06 15:54:53	Dear lcorrocher92, Thank you for registering for Google's Gemini model. You signed up on 2024-05-06 15:54:51.
	9	lucacorrocher@ucdconnect.com	2024-05-06 15:54:53	Dear lcorrocher92, Thank you for registering for Technology Innovation Institute's Falcon.7B model. You signed up
	10	bobwilliams@belvedere.ie	2024-05-06 15:54:53	Dear bwilliams18, Thank you for registering for OpenAI's GPT3.5 Turbo model. You signed up on 2024-05-06 15:54:
	11	bobwilliams@belvedere.ie	2024-05-06 15:54:53	Dear bwilliams18, Thank you for registering for Google's Gemini model. You signed up on 2024-05-06 15:54:51. You
	12	bobwilliams@belvedere.ie	2024-05-06 15:54:53	Dear bwilliams18, Thank you for registering for Technology Innovation Institute's Falcon.7B model. You signed up o
	13	gavincullen@gonzaga.ie	2024-05-06 15:54:53	Dear gcullen21, Thank you for registering for OpenAI's GPT3.5 Turbo model. You signed up on 2024-05-06 15:54:5

Figure 24. List of records for sendRegistrationEmails stored procedure.

Similar to the template prompt and the full prompt, a sample email can be implemented as seen below.



```sql
SET email_content = CONCAT(
    'Dear ', username, ',\n\n',
    'Thank you for registering for ', company, '\'s ', model_name, ' model.\n\n',
    'You signed up on ', DATE_FORMAT(registration_date, '%Y-%m-%d %H:%i:%s'), '.\n\n',
    'Your API Key is: ', api_key, '.\n\n',
    'You are using the ', model_name, ' model developed by ', company, '.\n\n',
    'You are a ', role, ' so you have been granted access to the ',
    CASE
        WHEN role IN ('pupil', 'undergrad student', 'postgrad student') THEN 'Academic package.'
        WHEN role IN ('teacher', 'lecturer', 'researcher') THEN 'Premium package.'
        ELSE 'Standard package.'
    END,
    '\n\nPlease click on the link to confirm your account!',
    '\n\nBest regards,\n\n',
    ceo, '\n',
    'CEO, ', company, '.');
```

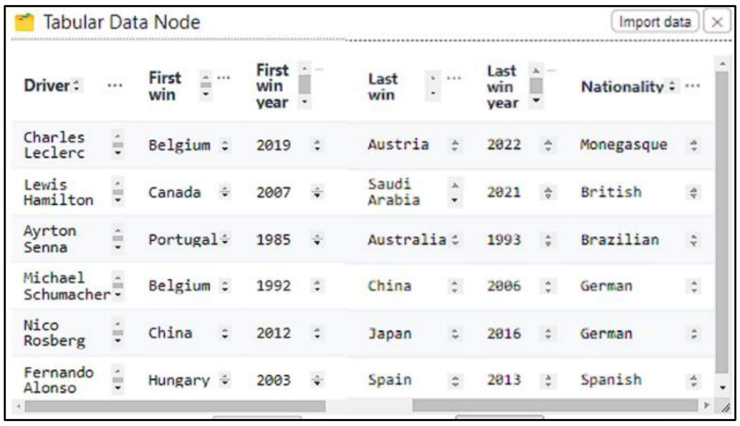
```
Dear lcorrocher92,

Thank you for registering for Google's Gemini model.
You signed up on 2024-05-06 15:54:51.

Your API Key is: e2b3108f-6713-4b90-8bb3-37fa524e2c6.

You are using the Gemini model developed by Google.

You are a postgrad student so you have been granted access to the Academic package.

Please click on the link to confirm your account!

Best regards,

Sundar Pichai
CEO, Google.
```

Figure 25. Send Bulk Registration Emails stored procedure and implemented email content.

Functions

getDriverInfo

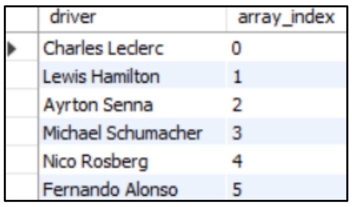
Given that the RDBMS will like scale to a big data architecture, overloaded the database with table records of single VARCHAR responses seemed inefficient if vertically scaled [12]. It was decided that the best approach for information to query should be encapsulated in a JSON object. This can be seen in the input node and tabular data node. JSON objects don't have a requirement maximum length which made this a very useful data storage medium for ChainForge RDBMS. Instead of multiple table records, a JSON object with the same number of arrays seemed more efficient.



Driver	First win	First win year	Last win	Last win year	Nationality
Charles Leclerc	Belgium	2019	Austria	2022	Monegasque
Lewis Hamilton	Canada	2007	Saudi Arabia	2021	British
Ayrton Senna	Portugal	1985	Australia	1993	Brazilian
Michael Schumacher	Belgium	1992	China	2006	German
Nico Rosberg	China	2012	Japan	2016	German
Fernando Alonso	Hungary	2003	Spain	2013	Spanish

Figure 26. Tabular data node for workflow. getDriverInfo uses the attributes as parameters to query the JSON object.

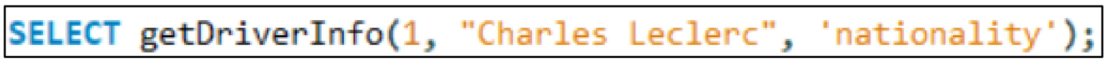
A driver name was used to map the driver to the JSON index. In the context of this demo database design, the JSON object was implemented with storing key-value pairs for formula 1 drivers with the following keys: driver, first win (country of), first win year, last win (country of), last win year, nationality). The list of keys could be enhanced in the future to store a more comprehensive driver dataset.



driver	array_index
Charles Leclerc	0
Lewis Hamilton	1
Ayrton Senna	2
Michael Schumacher	3
Nico Rosberg	4
Fernando Alonso	5

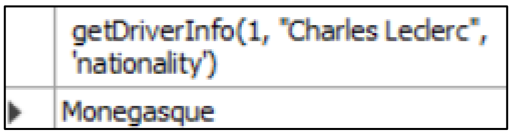
Figure 27. Driver index mapping table records

In order to classify the responses in the simple and code evaluator nodes, the responses from the LLMs were compared to the corresponding queries extracted from the JSON object. The getDriverInfo function allows the query to extract the record based on the tabular data node id (1 in this case), array index key (driver) and the desired key. An example implementation for this would be to say, query the nationality of Charles Leclerc. Consider the following correct query:



```
SELECT getDriverInfo(1, "Charles Leclerc", 'nationality');
```

Figure 28. Implemented getDriverInfo query.



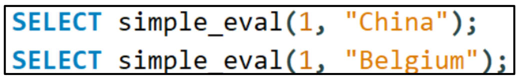
getDriverInfo(1, "Charles Leclerc", 'nationality')
Monegasque

Figure 29. Result of query. Returns the corresponding value.

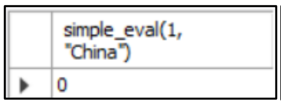
simple_eval

This function performs a Boolean comparison match between a response_node.response with varchar driver_info, returning 1 if they are equal and 0 otherwise. It depends on a unique response_id (primary superkey).

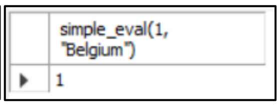
For example, knowing that driver Charles Leclerc's first race win was in the country "*Belgium*" and the response_id the model generated, we can call the simple_eval function with the correct and incorrect value and view the results:



```
SELECT simple_eval(1, "China");
SELECT simple_eval(1, "Belgium");
```



simple_eval(1, "China")
0

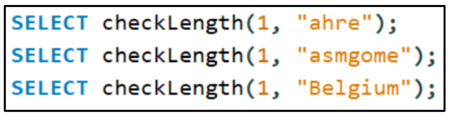


simple_eval(1, "Belgium")
1

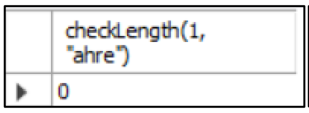
checkLength

Similarly, the checkLength function performs a Boolean comparison with the length of each varchar as criterion. If the lengths are equal, the result returns 1 and 0 otherwise. Although not the best metric to compare if two responses are equal, it's what the code evaluator uses and acts as a loose check.

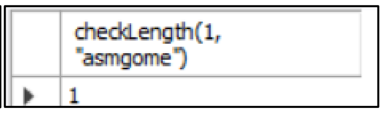
An example is seen below, given the same response_id = 1, where the correct response should be "*Belgium*", length 7:



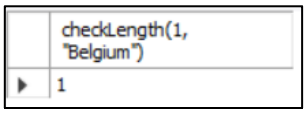
```sql
SELECT checkLength(1, "ahre");
SELECT checkLength(1, "asmgome");
SELECT checkLength(1, "Belgium");
```



```
checkLength(1,
"ahre")
0
```



```
checkLength(1,
"asmgome")
1
```



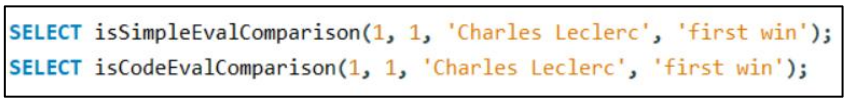
```
checkLength(1,
"Belgium")
1
```

A response could be an entire other word but if it's the same length, it passes this classification. Note that "*ahre*" fails but "*asmgome*" returns true as it has the same length as "*Belgium*". In future, evaluators should enforce more stringent conditions to compare the two responses.

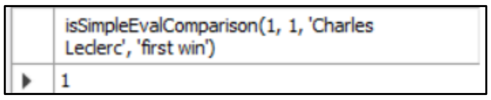
isSimpleEvalComparison/ isCodeEvalComparison

These functions build on the getDriverInfo and either the simple_eval or checkLength functions by embedding them together. isSimpleEvalComparison compares the value of the entered key with the model response of the entered response_node_id. The function takes response_node_id, tabular_data_node_id, driver and key_param as inputs. The getDriverInfo returns a driver_info variable, which is either passed into simple_eval/checkLength functions to return a Boolean result. This is how the database can compare the model's response it wants to test with the actual value from the tabular data node record.

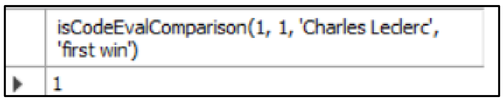
An implementation can be seen below:



```sql
SELECT isSimpleEvalComparison(1, 1, 'Charles Leclerc', 'first win');
SELECT isCodeEvalComparison(1, 1, 'Charles Leclerc', 'first win');
```



```
isSimpleEvalComparison(1, 1, 'Charles
Leclerc', 'first win')
1
```



```
isCodeEvalComparison(1, 1, 'Charles Leclerc',
'first win')
1
```

Given the answer for respons_node_id 1 is "*Belgium*", if either the response_node_id wasn't equal to 1 or the driver's name or key parameter were different, the isSimpleEvalComparison function would return false. The isCodeEvalComparison enforces a less strict classification check, whereby it calls the checkLength function.

isDefaultLLMEvalComparison

The isDefaultLLMEvalComparison function is very similar to the other two above, but it relates to the LLM scorer evaluator node. The function takes the response_node_id and llm_scorer_response_id as inputs. It compares the LLM scorer prompt response answer with the response node's answer based on a strict match. The response value must be equal to the LLM scorer response value. If they match, the function returns true, otherwise it returns false.

Given that we have discovered that response_node_id 1 is model_id 1 (GPT3.5 Turbo's) response attempt to the question "*What country did the F1 driver Charles Leclerc win his first Grand Prix?*" is "*Belgium*", we can compare this to when llm_scorer_response.response_id = 1, i.e. "*Belgium*".

This function should return true:

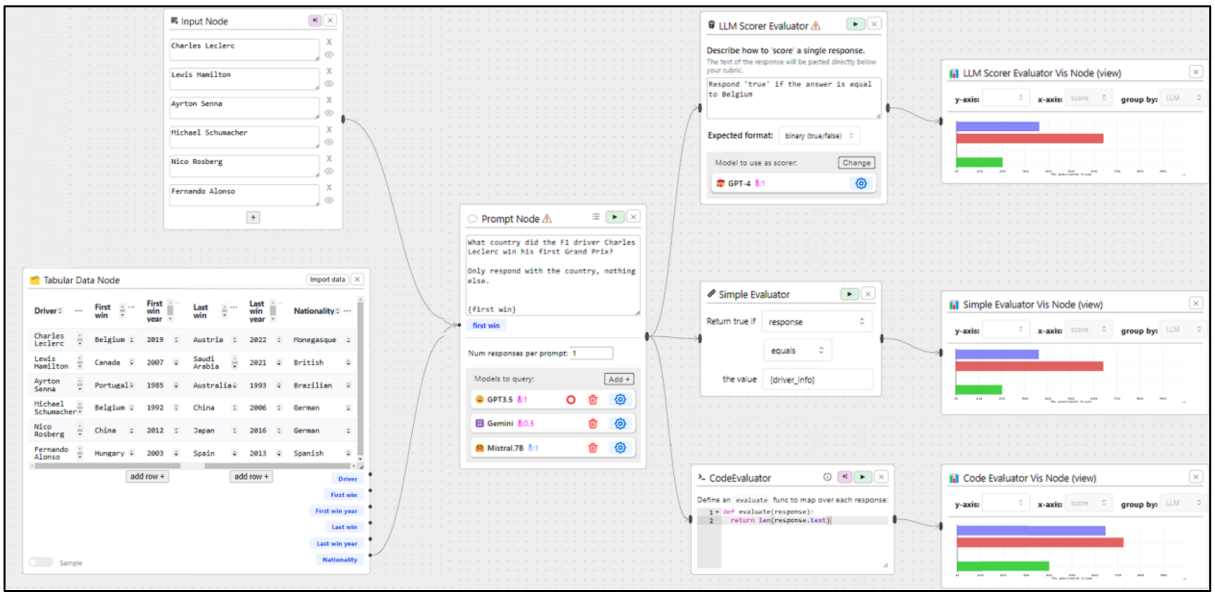
```sql
SELECT isDefaultLLMEvalComparison(1, 1);
```

	isSimpleEvalComparison(1, 1, 'Charles Leclerc', 'first win')
▶	1

Any unequal responses should return false.

6. Example Queries: Database In Action

Walkthrough of Workflow



Input Node:

```
Charles Leclerc
Lewis Hamilton
Ayrton Senna
Michael Schumacher
Nico Rosberg
Fernando Alonso
```

Tabular Data Node:

Driver	First win	First win year	Last win	Last win year	Nationality
Charles Leclerc	Belgium	2019	Austria	2022	Monegasque
Lewis Hamilton	Canada	2007	Saudi Arabia	2021	British
Ayrton Senna	Portugal	1985	Australia	1993	Brazilian
Michael Schumacher	Belgium	1992	China	2006	German
Nico Rosberg	China	2012	Japan	2016	German
Fernando Alonso	Hungary	2003	Spain	2013	Spanish

Prompt Node:

```
What country did the F1 driver Charles Leclerc win his first Grand Prix?

Only respond with the country, nothing else.

{first win}
```

Num responses per prompt: 1

Models to query: GPT3.5, Gemini, Mistral.7B

LLM Scorer Evaluator: Describe how to 'score' a single response. The text of the response will be pasted directly below your rubric.

```
Respond 'true' if the answer is equal to Belgium
```

Expected format: binary (true/false)

Model to use as scorer: GPT-4

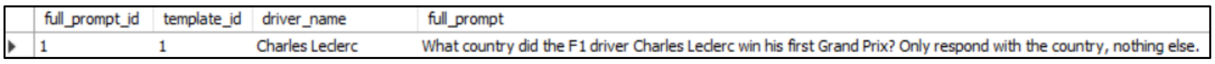
Simple Evaluator: Return true if response equals the value {driver_info}

CodeEvaluator: Define an evaluate func to map over each response:

```python
def evaluate(response):
  return len(response.text)
```

Figure 30. Charles Leclerc country first win (prompt_node.full_prompt_id = 1)

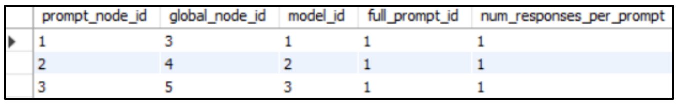
Luca (user_id = 1) want to know wants to test one particular chain id 1, with the prompt_id 1, mapping to full prompt"*What country did the F1 driver Charles Leclerc win his first Grand Prix?*"



full_prompt_id	template_id	driver_name	full_prompt
1	1	Charles Leclerc	What country did the F1 driver Charles Leclerc win his first Grand Prix? Only respond with the country, nothing else.

Figure 31. full_prompts.full prompt _id = 1

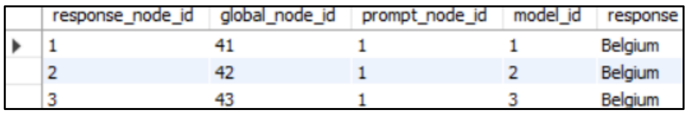
Querying the models GPT3.5 Turbo, Gemini and Falcon 7.B at entropy temperature 0 with one prompt response per model, Luca runs the prompt node. This induces the prompt node store this prompt node information into three records, one to be queried by each model.



prompt_node_id	global_node_id	model_id	full_prompt_id	num_responses_per_prompt
1	3	1	1	1
2	4	2	1	1
3	5	3	1	1

Figure 32. Prompt segregation into individual records destined for individual models.

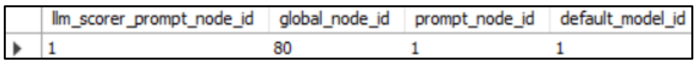
These prompt node records diverge and are sent to each LLM using the unique API key for the user and a response record for each corresponding prompt node is generated containing a response from each model. In this case, all three responses are "*Belgium*".



response_node_id	global_node_id	prompt_node_id	model_id	response
1	41	1	1	Belgium
2	42	1	2	Belgium
3	43	1	3	Belgium

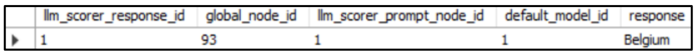
Figure 33. Response node records for corresponding prompt node ids.

At the same time, the prompt node is sent to the LLM scorer prompt node, which prompts the accurate default model with the same question and prompt_id. This model responds with an answer:



llm_scorer_prompt_node_id	global_node_id	prompt_node_id	default_model_id
1	80	1	1

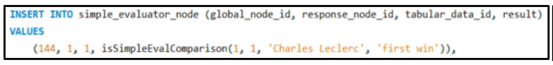
Figure 34. Prompt node destined for LLM default model (GPT4, temperature 0).



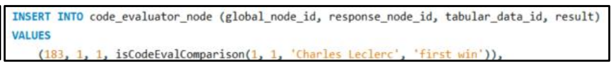
llm_scorer_response_id	global_node_id	llm_scorer_prompt_node_id	default_model_id	response
1	93	1	1	Belgium

Figure 35. LLM default model response.

The previous response_node responses are then queried against the tabular data with the tabular data node id 1, driver "*Charles Leclerc*", and key "*first win*" using the isSimpleEvalComparison and the isCodeEvalComparison, these methods both call the getDriverInfo method and the comparison functions simple_eval/checkLength to compare the tabular data value for first win attribute. In all comparison's the result is true, and the models have successfully generated the appropriate response to the prompt.

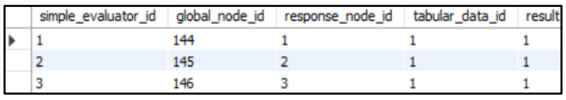


```
INSERT INTO simple_evaluator_node (global_node_id, response_node_id, tabular_data_id, result)
VALUES
    (144, 1, 1, isSimpleEvalComparison(1, 1, 'Charles Leclerc', 'first win')),
```

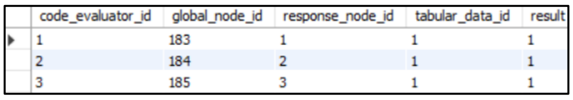


```
INSERT INTO code_evaluator_node (global_node_id, response_node_id, tabular_data_id, result)
VALUES
    (183, 1, 1, isCodeEvalComparison(1, 1, 'Charles Leclerc', 'first win')),
```

Figure 36. isSimpleEvalComparison called for simple evaluator. *Figure 37. isCodeEvalComparison* called for code evaluator.



simple_evaluator_id	global_node_id	response_node_id	tabular_data_id	result
1	144	1	1	1
2	145	2	1	1
3	146	3	1	1

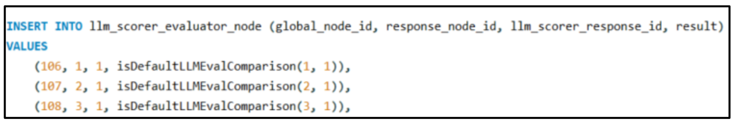


code_evaluator_id	global_node_id	response_node_id	tabular_data_id	result
1	183	1	1	1
2	184	2	1	1
3	185	3	1	1

Figure 36. Resulting code evaluator node records.

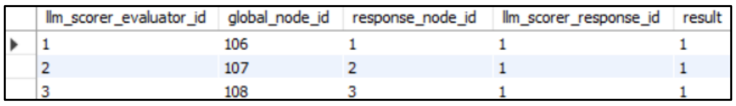
Figure 37. Resulting simple evaluator node records.

The LLM scorer evaluator node also calls the isDefaultLLMEvalComparison function. All the models have successfully passed the Boolean response comparison.



```
INSERT INTO llm_scorer_evaluator_node (global_node_id, response_node_id, llm_scorer_response_id, result)
VALUES
    (106, 1, 1, isDefaultLLMEvalComparison(1, 1)),
    (107, 2, 1, isDefaultLLMEvalComparison(2, 1)),
    (108, 3, 1, isDefaultLLMEvalComparison(3, 1)),
```

Figure 38. isDefaultLLMEvalComparison called for LLM evaluator.



llm_scorer_evaluator_id	global_node_id	response_node_id	llm_scorer_response_id	result
1	106	1	1	1
2	107	2	1	1
3	108	3	1	1

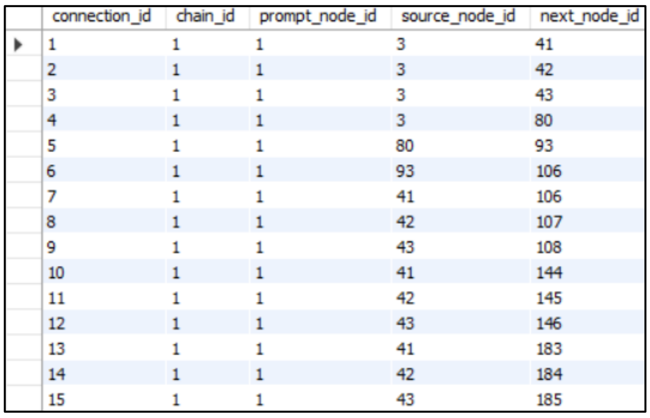
Figure 39. Resulting LLM evaluator node records.

The chain node connections table shows the flow history of all the records, based on their global node id. Each node travels from a source node to a destination record. Observe the converging and diverging nature of the workflow. Node 3 (prompt record) travel to 41, 42 and 43 (response records). Later 41, 42, and 43 diverge again.

Node 41 (response record with GPT 3.5 Turbo) diverges to 106, 144, 183 (simple/code/LLM evaluator nodes).

Node 42 (response record with Gemimi) diverges to 107 to 145, 184.

Node 43 (response record with Falcon 7.B) diverges to 108, 146, 185 respectively.



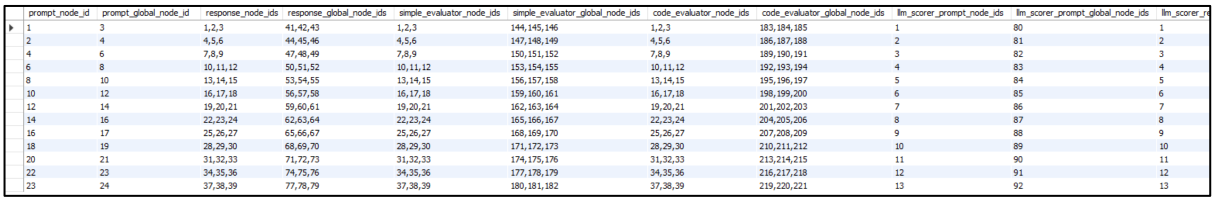
connection_id	chain_id	prompt_node_id	source_node_id	next_node_id
1	1	1	3	41
2	1	1	3	42
3	1	1	3	43
4	1	1	3	80
5	1	1	80	93
6	1	1	93	106
7	1	1	41	106
8	1	1	42	107
9	1	1	43	108
10	1	1	41	144
11	1	1	42	145
12	1	1	43	146
13	1	1	41	183
14	1	1	42	184
15	1	1	43	185

Figure 40. Chain node connection edge flow. Represents direction of chain.

Although this is only a granular walkthrough for one specific prompt id and three LLMs, this RDBMS design is scalable for thousands of prompt nodes and many more LLMs. The current populated RDBMS contains 4 template prompts, one JSON object with 6 records, totalling to 24 full prompts. Only 13 have been put through the workflow, resulting in 12 x 3 x 3 = 117 evaluated nodes, delivering accurate insights at the visualisation views.

Prompt Node Associations

Below is a query which shows a sample set of records which show which specific child node ids and global node ids are associated for each prompt. This is useful for identifying the path of each chain.

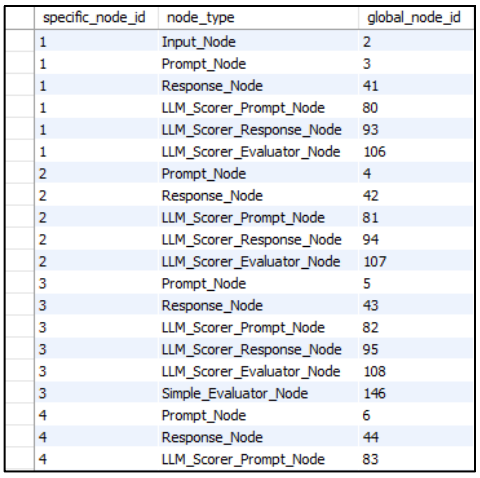


prompt_node_id	prompt_global_node_id	response_node_ids	response_global_node_ids	simple_evaluator_node_ids	simple_evaluator_global_node_ids	code_evaluator_node_ids	code_evaluator_global_node_ids	llm_scorer_prompt_node_ids	llm_scorer_prompt_global_node_ids	llm_scorer_re
1	3	1,2,3	41,42,43	1,2,3	144,145,146	1,2,3	183,184,185	1	80	1
2	4	4,5,6	44,45,46	4,5,6	147,148,149	4,5,6	186,187,188	2	81	2
4	6	7,8,9	47,48,49	7,8,9	150,151,152	7,8,9	189,190,191	3	82	3
6	8	10,11,12	50,51,52	10,11,12	153,154,155	10,11,12	192,193,194	4	83	4
8	10	13,14,15	53,54,55	13,14,15	156,157,158	13,14,15	195,196,197	5	84	5
10	12	16,17,18	56,57,58	16,17,18	159,160,161	16,17,18	198,199,200	6	85	6
12	14	19,20,21	59,60,61	19,20,21	162,163,164	19,20,21	201,202,203	7	86	7
14	16	22,23,24	62,63,64	22,23,24	165,166,167	22,23,24	204,205,206	8	87	8
16	17	25,26,27	65,66,67	25,26,27	168,169,170	25,26,27	207,208,209	9	88	9
18	19	28,29,30	68,69,70	28,29,30	171,172,173	28,29,30	210,211,212	10	89	10
20	21	31,32,33	71,72,73	31,32,33	174,175,176	31,32,33	213,214,215	11	90	11
22	23	34,35,36	74,75,76	34,35,36	177,178,179	34,35,36	216,217,218	12	91	12
23	24	37,38,39	77,78,79	37,38,39	180,181,182	37,38,39	219,220,221	13	92	13

Figure 41. Prompt node associations - overview of child associations -ted to individual prompt.

Specific Child Node to Global Node

If the node ids become overwhelming to map specific ids to their type and their global node ids, the below query simplifies this issue:

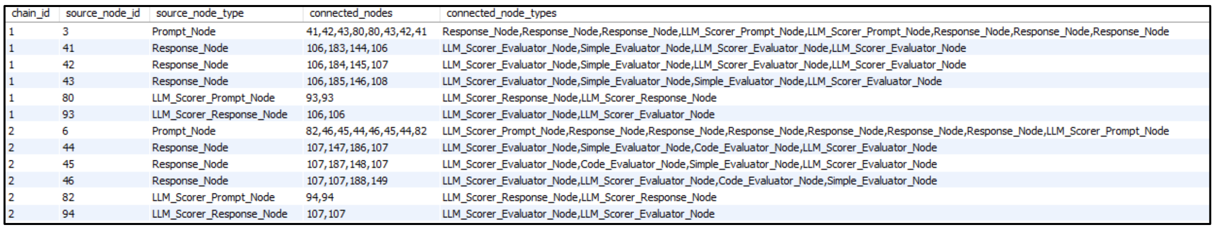


specific_node_id	node_type	global_node_id
1	Input_Node	2
1	Prompt_Node	3
1	Response_Node	41
1	LLM_Scorer_Prompt_Node	80
1	LLM_Scorer_Response_Node	93
1	LLM_Scorer_Evaluator_Node	106
2	Prompt_Node	4
2	Response_Node	42
2	LLM_Scorer_Prompt_Node	81
2	LLM_Scorer_Response_Node	94
2	LLM_Scorer_Evaluator_Node	107
3	Prompt_Node	5
3	Response_Node	43
3	LLM_Scorer_Prompt_Node	82
3	LLM_Scorer_Response_Node	95
3	LLM_Scorer_Evaluator_Node	108
3	Simple_Evaluator_Node	146
4	Prompt_Node	6
4	Response_Node	44
4	LLM_Scorer_Prompt_Node	83

Figure 42. Child Node to Global Node - shows the global node id of each child node id.

Adjacency List Graph

Below is a query which represents an adjacency list graph-like structure of the flow of nodes in a specific chain. It shows which nodes are connected to chain id 1.



chain_id	source_node_id	source_node_type	connected_nodes	connected_node_types
1	3	Prompt_Node	41,42,43,80,80,43,42,41	Response_Node,Response_Node,Response_Node,LLM_Scorer_Prompt_Node,LLM_Scorer_Prompt_Node,Response_Node,Response_Node,Response_Node
1	41	Response_Node	106,183,144,106	LLM_Scorer_Evaluator_Node,Simple_Evaluator_Node,LLM_Scorer_Evaluator_Node,LLM_Scorer_Evaluator_Node
1	42	Response_Node	106,184,145,107	LLM_Scorer_Evaluator_Node,Simple_Evaluator_Node,LLM_Scorer_Evaluator_Node,LLM_Scorer_Evaluator_Node
1	43	Response_Node	106,185,146,108	LLM_Scorer_Evaluator_Node,Simple_Evaluator_Node,Simple_Evaluator_Node,LLM_Scorer_Evaluator_Node
1	80	LLM_Scorer_Prompt_Node	93,93	LLM_Scorer_Response_Node,LLM_Scorer_Response_Node
1	93	LLM_Scorer_Response_Node	106,106	LLM_Scorer_Evaluator_Node,LLM_Scorer_Evaluator_Node
2	6	Prompt_Node	82,46,45,44,46,45,44,82	LLM_Scorer_Prompt_Node,Response_Node,Response_Node,Response_Node,Response_Node,Response_Node,Response_Node,LLM_Scorer_Prompt_Node
2	44	Response_Node	107,147,186,107	LLM_Scorer_Evaluator_Node,Simple_Evaluator_Node,Code_Evaluator_Node,LLM_Scorer_Evaluator_Node
2	45	Response_Node	107,187,148,107	LLM_Scorer_Evaluator_Node,Code_Evaluator_Node,Simple_Evaluator_Node,LLM_Scorer_Evaluator_Node
2	46	Response_Node	107,107,188,149	LLM_Scorer_Evaluator_Node,LLM_Scorer_Evaluator_Node,Code_Evaluator_Node,Simple_Evaluator_Node
2	82	LLM_Scorer_Prompt_Node	94,94	LLM_Scorer_Response_Node,LLM_Scorer_Response_Node
2	94	LLM_Scorer_Response_Node	107,107	LLM_Scorer_Evaluator_Node,LLM_Scorer_Evaluator_Node

Figure 43. Adjacency list graph representation of nodes in chain id.

Projected Accuracy of Model Performance

The below query groups the average model performance accuracy over all evaluator nodes. It was show that Gemini has the highest accuracy, whereas OpenAI's GPT 3.5 Turbo and TII's Falcon 7.B diverged from the true responses.

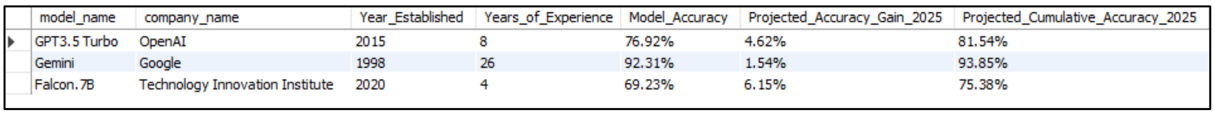
Generally, there is an inverse relationship between the gains made by a company on a machine learning LLM and the number of years they have been working on the model. Model learning rate generally increases exponentially at the begging and then begins to decay. Assuming the year of the company founding correlates to the beginning of model development, a mathematical model was proposed which attempts to project the accuracy for 2025. It is denoted as:

$$\alpha = 0.2, D = Decay\ factor, E = Error\ rate$$

$$Projected\ Accuracy\ Gain\ Next\ Year = \alpha D(100 - E)$$

$$where\ E = \left(\frac{Count\ of\ incorrect\ predictions}{Total\ count\ of\ predictions}\right) x\ 100, \quad D = 0.5^{\frac{1}{(Years\ of\ Experience+1)}}$$

The projected model performance cumulative accuracy for 2025 sums the current model accuracy with the projected accuracy gain.



	model_name	company_name	Year_Established	Years_of_Experience	Model_Accuracy	Projected_Accuracy_Gain_2025	Projected_Cumulative_Accuracy_2025
▶	GPT3.5 Turbo	OpenAI	2015	8	76.92%	4.62%	81.54%
	Gemini	Google	1998	26	92.31%	1.54%	93.85%
	Falcon.7B	Technology Innovation Institute	2020	4	69.23%	6.15%	75.38%

Figure 44. Projected accuracy query of model performance.

The results outline that if TII began learning in 2020, their projected annual gain for 2025 will be the 6.15%, the highest, and they are expected to slowly catch up with its competitors. In comparison, Google's Gemini is projected to gain 1.54% model performance accuracy and OpenAI's GPT 3.5 Turbo a 4.62% gain.

7. Conclusions

The proposed design for the RDBMS in the ChainForge prompt engineering environment demonstrates a comprehensive approach to achieving efficiency, robustness, and flexibility, making it a powerful evaluation tool. The design goals and motivations behind the system have been carefully considered to address key aspects of model design, prompt templates, streamlined evaluation, flexibility, and customization.

One of the notable features of the design is its accommodation of JSON formats for data storage, utilizing Tabular Data Nodes and Input Nodes to store attribute values and generate full prompts from prompt templates. This approach facilitates the querying of prompts by multiple large language models, with each response evaluated differently based on specified parameters. The incorporation of multiple evaluators, including the LLM Scorer Evaluator with temperature variation, enhances the robustness and generalizability of model performance metrics, particularly when tested on diverse datasets.

The auto generation of multiple full prompts from template prompts, based on the number of attributes in a workflow, contributes to a larger dataset of prompt responses, hence improving model accuracy and reducing bias. Node convergence and divergence enable parallel processing and feature combination, allowing for efficient handling of data layers and visualization of results for comparison.

In summary, the RDBMS design offers variability in Prompt Nodes, divergence of responses from each model, and convergence of results for analysis and visualization. This platform provides a powerful means to diagnose model flaws, stress test models, and analyse their limitations. With its robust design and thoughtful implementation, the system is poised to fulfil its objectives effectively within the ChainForge prompt engineering environment. Ongoing review and refinement will further enhance its performance and utility in practice.

Reference

[1] B. H. M. h. A. S. Markus Binder, “Global reconstruction of language models with linguistic rules – Explainable AI for online consumer reviews,” vol. 32, 2022.

[2] Y. S. H. Q. X. H. X. Q. Tianxiang Sun, “Black-Box Tuning for Language-Model-as-a-Service,” in *Proceedings of the 39th International Conference on Machine Learning,* , 2022.

[3] A. I. C. P. e. al., “ChainForge: A Visual Toolkit for Prompt Engineering and LLM Hypothesis Testing,” 2023.

[4] U. Ç. Michael Stonebraker, ““One Size Fits All”: An Idea Whose Time Has Come and Gone,” in *International Conference on Data Engineering (ICDE)*, 2005.

[5] D. Sullivan, NoSQL for Mere Mortals, Addison-Wesley Professional.

[6] A. Pavlo, “The Official Ten-Year Retrospective of NewSQL”.

[7] N. S. Elmasri Ramez, Fundamentals Of Database System, Pearson, 2015.

[8] “MySQL 8.0 Reference Manual,” 2024. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/.

[9] W. Kent, “A Simple Guide to Five Normal Forms in Relational Database Theory,” 1982.

[10] E. F. Codd, “Information Retrieval A Relational Model of Data for Large Shared Data Banks,” vol. 3, no. 6, 1979.

[11] “Consistency Models,” Jepsen LLC, 2016. [Online]. Available: https://jepsen.io/consistency.

[12] M. Kleppmann, Designing Data-Intensive Applications, O'Reilly Media, Inc., 2017.

[13] W. Vogels, “Eventually consistent,” *Communications of the ACM,* vol. 52, 2009.

[14] E. Brewer, “CAP Twelve Years Later: How the "Rules" Have Changed,” *IEEE Computer Society,* 2023.

[15] J. G. Daivid Dewitt, “Parallel Database Systems: The Future of High Performance Database Systems,” *Communications Of The ACM,* vol. 35, no. 6, 1992.

[16] G. K. W. A. e. al., “Data management in cloud environments: NoSQL and NewSQL data stores,” *Journal of Cloud Computing: Advances, Systems and Applications,* 2013.

[17] N. H. P. H. e. a. Michael Stonebraker, “Proceedings of the 33rd international conference on Very large data bases,” 2007.

[18] M. A. Andrew Pavlo, “What’s Really New with NewSQL?,” *SIGMOD,* vol. 45, no. 2, 2016.