DataChat: An Intuitive and Collaborative Data Analytics Platform

Rogers Jeffrey Leo John
DataChat Inc.
Madison, WI, USA
rogers@datachat.ai

Dylan Bacon
DataChat Inc.
Madison, WI, USA
dylan@datachat.ai

Junda Chen
DataChat Inc.
Madison, WI, USA
junda@datachat.ai

Ushmal Ramesh
DataChat Inc.
Madison, WI, USA
ushmal@datachat.ai

Jiatong Li
DataChat Inc.
Madison, WI, USA
jiatong@datachat.ai

Deepan Das
DataChat Inc.
Madison, WI, USA
deepan.das@datachat.ai

Robert Claus
DataChat Inc.
Madison, WI, USA
robert@datachat.ai

Amos Kendall
DataChat Inc.
Madison, WI, USA
amos@datachat.ai

Jignesh M. Patel
DataChat Inc.
Madison, WI, USA
jignesh@datachat.ai

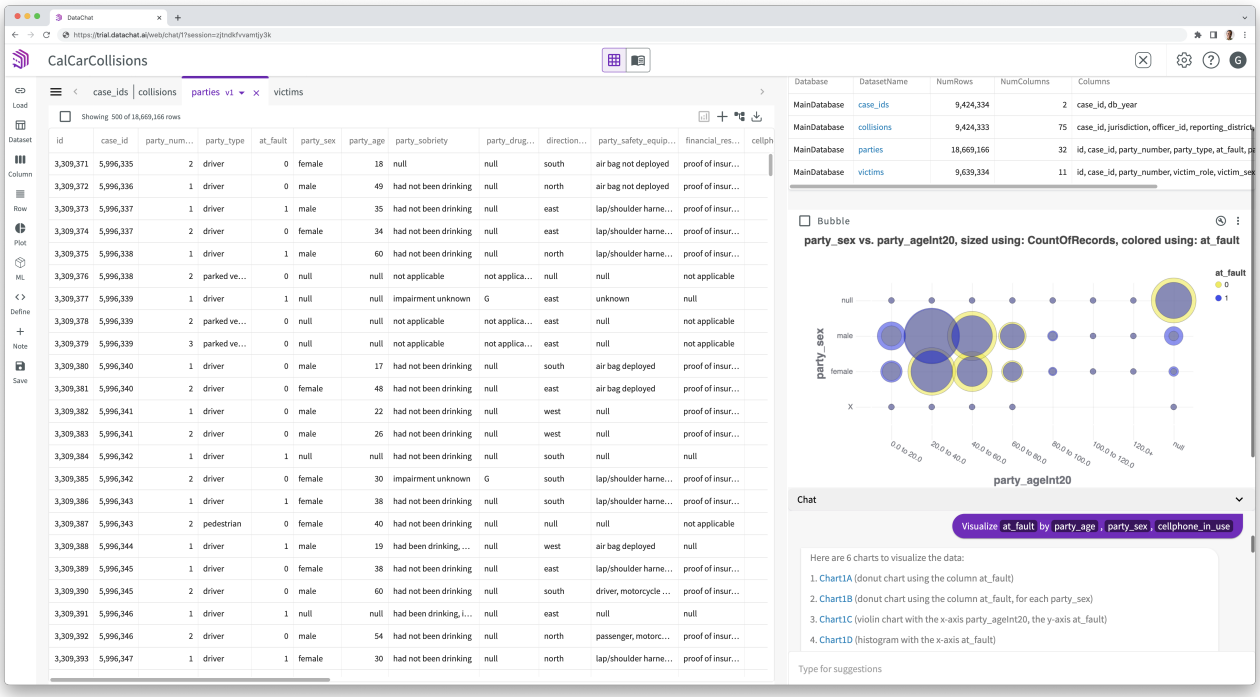


Figure 1: An Interactive DataChat Session.

ABSTRACT

Enterprises invest in data platforms with the aim of extracting meaningful information through analytics. Typically, experts create analytics pipelines that feed into dashboards and provide answers to predetermined questions. This approach makes analytics a spectator sport for most people and introduces operational bottlenecks to leveraging those investments. To improve the value derived from data, many organizations are opting to open up their data assets and allow access to a wider range of users. However, using programming languages such as SQL and Python for analytics can be difficult for most enterprise users. DataChat provides a simplified data science approach that is intuitive, powerful, and accessible to *all* data users. The platform is built on a library of data functions that are cleanly

SIGMOD-Companion '23, June 18–23, 2023, Seattle, WA, USA

ACM ISBN 978-1-4503-9507-6/23/06.
https://doi.org/10.1145/3555041.3589678

abstracted to maximize efficiency and ease of use while maintaining a rich suite of tools necessary for data science. With these functions, users can create data analysis pipelines by using a simple point-and-click interface in a spreadsheet view or by using natural English interfaces. Modern sharing and collaboration features are central to all aspects of the platform, allowing teams to easily bridge expertise gaps. A deeper understanding of results is facilitated by providing automatically-generated English explanations of how they were derived. By enhancing these aspects of data science and human-to-human communication, the platform addresses the needs that many organizations are encountering as their analytics needs mature.

CCS CONCEPTS

• **Information systems → Data analytics**.

KEYWORDS

Analytics, Data Science, Machine Learning, Generative AI

ACM Reference Format:

Rogers Jeffrey Leo John, Dylan Bacon, Junda Chen, Ushmal Ramesh, Jiatong Li, Deepan Das, Robert Claus, Amos Kendall, and Jignesh M. Patel. 2023. DataChat: An Intuitive and Collaborative Data Analytics Platform. In *Companion of the 2023 International Conference on Management of Data (SIGMOD-Companion '23), June 18–23, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3555041.3589678

1 INTRODUCTION

Enterprises are constantly seeking a competitive edge, and their data infrastructure is a crucial tool in achieving this goal. While collecting data in a warehouse or lakehouse is important, it's secondary to the primary objective of extracting meaningful insights. However, existing methodologies tend to be time-consuming and demand specialized programming skills from dedicated teams, resulting in process-heavy organizational structures.

In the early days of data analytics, "insights" were largely defined as historical reporting. At the end of the last century, Crystal Reports [19, 37] was a state-of-the-art reporting tool, but it had only primitive capabilities to support live enterprise dashboards. At the turn of the century, the Polaris [31, 32] project ushered in a new approach to business analytics that led to the development of Tableau [30]. Enterprise dashboards, which allowed organizations to track key performance indicators (KPIs) to support day-to-day operations, became common. Tableau's success prompted the emergence of other dashboard products, such as Power BI [22], Qlik [25], and Domo [8]. The ubiquity of such needs has also led many data platform vendors to integrate limited dashboarding capabilities directly into their core offering.

Over the past decade, a new category of analytics tools has emerged, such as Looker [18], Thoughtspot [33], and more recently dbt metrics [7]. These tools leverage the benefits of an *explicit* semantic layer that contains curated definitions, such as `revenue` being the sum of the product of `price` and `(1 - Discount)` for all relevant orders. In the early generations of dashboard tools, the semantic layer was not explicitly abstracted, making it challenging to manage these semantic definitions. Many of these new tools also experiment with novel modalities for user interactions, such as creating charts based on phrases defined in the semantic layer. Natural language interfaces in particular have grown in popularity and have even been adopted by more traditional dashboard platforms. Another example is pairing a user's request for a chart with generated textual descriptions of key insights. (Section 4 discusses this and other approaches to translating user intent to insights.)

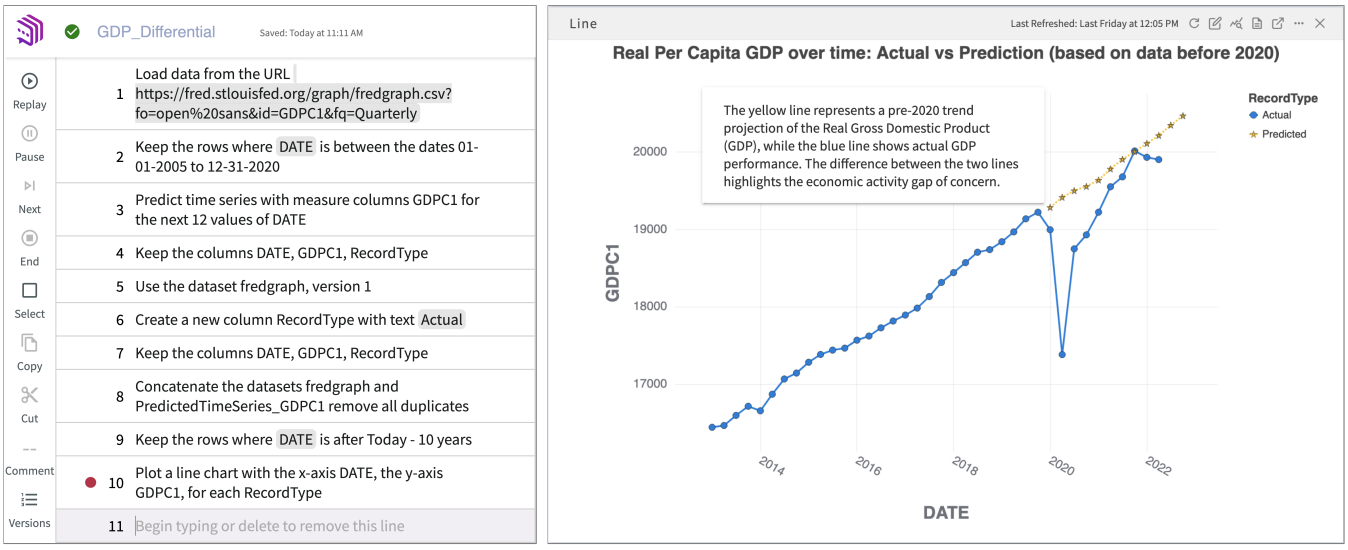
However, a significant challenge still exists. While dashboarding tools work well for telling users *what* is happening to a KPI, they have limited capabilities to allow a user to carry out a root-cause analysis to determine *why* the KPI looks that way. While some startups such as Sisu [6] explicitly address this challenge, most tools have limited capabilities for generalizing such exploration. Instead, they provide tools such as drill-up/down, along a pre-defined, semantic hierarchy or allow users to change basic chart properties to attempt to visually identify the root cause(s).

There are two common methods for unrestricted data exploration. The first is the long-standing and omnipresent Excel [9, 21]. Excel can handle many data exploration and simple analytics tasks, and users often download data from dashboards to analyze it in Excel. However, there are three major limitations: a) Excel has data limits, such as a maximum of one million rows [20], b) Excel lacks modern, ML-based methods for exploring and identifying patterns in data, and c) repeating the analysis at a later time is a manual process. Google Sheets and other spreadsheet tools have similar constraints, albeit with different limits. Newer tools like Sigma Computing [10] offer a spreadsheet view of data, which is also a topic of interest in the research community [1, 4, 26], but they primarily focus on visualization and dashboarding capabilities with limited coverage of data science and machine learning functionality.

The second dominant approach for unconstrained data exploration is using modern data science tools like Jupyter Notebook [23], or its modernized versions like Google Colab [11] or Hex Notebooks [13]. However, this approach requires the user to be a programmer, with detailed knowledge of languages like Python, SQL, and R. As a result, this approach can be challenging for enterprise data users who are not data scientists.

The central idea behind DataChat is to provide an easy-to-use, natural, and intuitive platform that enables users to apply data science principles on large datasets without having to master the mechanics of data science programming. These mechanics include not only a language like Python but also its various subsystems, such as Pandas, numpy, and other machine learning (ML) libraries. DataChat aims to separate the need to learn data science coding and library interface mechanics from the ability to use data science principles, which involve logically utilizing essential building blocks to solve data problems.

To achieve this goal, DataChat offers a spreadsheet-like interface that is simple, familiar, and free of the limitations of spreadsheets. Figure 1 shows a screenshot of the DataChat user interface. Users can connect to databases (or queries/views within them), CSV files, or a combination of both and perform analytics using a set of high-level data science functions called *skills*. DataChat simplifies data science functions into a set of around 50 high-level skills, which users can easily understand and use to express their intent. These skills provide a declarative abstraction for the most common data science functions in a natural way, and can be invoked through simple UI gestures. For example, a user can right-click on a value to keep all the rows that have the same value (similar to a selection



(a) DataChat recipe in an editor. **(b) The corresponding chart artifact.**

Figure 2: An editable DataChat recipe in GEL (a), corresponding to the chart artifact (b). The recipe is accessible from the artifact menu and takes the user to an editor.

predicate in SQL). These skills cover common data analysis functions, including data visualization, ML-based discovery, ML-based insights generation, common data wrangling tasks, common SQL tasks, and various collaboration functions.

The DataChat platform enables users to generate different types of *artifacts*, including reports, charts, ML models, and ML-based explanations. As users interact with the system, the platform automatically generates a *recipe* that captures the sequence of steps in the analysis. Each artifact is accompanied by the recipe that explains the steps taken to produce it. The recipe is typically shown to users in a controlled natural language, called Guided English Language (GEL), as shown in Figure 2. GEL is a declarative programming language with English syntax and provides an intuitive way for a user to directly interface the individual steps in a recipe. Users can approach a recipe as GEL "code" and use a built-in editor with debugging capabilities to step through each line (examining the output at each step if needed), and make edits directly to the lines/steps expressed in GEL as needed.

DataChat also offers built-in collaboration capabilities to support analytics as a group activity. Users can co-create an artifact (and the underlying recipe) using a live collaborative session similar to pair-programming in traditional programming environments. Any artifact can be shared with other users, and because each artifact has a recipe, the recipient can "continue the conversation" and explore the results further. This is possible even across significantly technically-diverse teams because recipes are presented in the natural language syntax of GEL rather than requiring specialized technical skills.

Recipes in DataChat can be created in multiple ways. One way is through a series of direct user requests in the user interface. Another way is by typing a natural language sentence into a chat box, which the system converts to discrete intents using a Large Language Model (LLM). Prompt engineering methods and database schema hints work together to generate a target response. DataChat GEL is one of the languages used for the generated response code, which makes it easier for humans to edit the response if additional changes are needed. For simpler use cases, a phrase-based translational layer is used, where the input text is a combination of predefined phrases in a semantic layer. This simpler approach is often preferred for use cases where the user's questions are more structured and driven by terms that can be easily encoded in the semantic layer. The advantage of this approach is higher accuracy in translating the intent to the response.

Overall, DataChat presents a new, intuitive, and collaborative way for enterprises to empower a broad range of data users to carry out ad hoc analysis, share their creations with others in the organization, and enable anyone to understand and reproduce the shared artifacts.

The remainder of this paper is organized as follows. In the next section, we present an overview of the DataChat platform. Section 3 highlights two key mechanisms that makes working with large cloud databases efficient both from the human and cloud-cost perspectives. Section 4 outlines the key methods that are used to generate DataChat recipes from natural language, which includes both a phrase-based approach and an LLM-based approach. Finally, Section 5 contains our concluding remarks.

2 THE DATACHAT PLATFORM

In this section, we describe the key components of the DataChat platform.

2.1 Skills

To provide an intuitive way for users to complete deep data science workflows, the platform provides carefully curated primitive operations within that space. These primitive operations, or *skills*, provide the building blocks for deeper data analysis within DataChat.

Skills differ from most programming languages where fundamental operations reflect computational or program flow primitives, not logical steps. For example, most tabular data operations do not require the concept of a loop, whereas traditional programming languages fundamentally depend on it. Libraries built on such languages, though built specifically for data science, still inherit the fundamental organization of those languages, resulting in a mix of declarative and procedural operations.

Even data query languages like SQL still diverge from the way data users approach problems. Data users approach real-world problems in sequential steps that are often highly diverse as they experiment with different approaches to the problem. Query languages, however, focus on single, parameterized requests. This makes investigations difficult if they require query iteration, multiple steps, or wildly different domain tools.

Skills in DataChat cover a wide range of use-cases that users frequently switch between when creating and iterating over data science workflows. By allowing this back-and-forth interaction within the same platform, users can have a seamless experience, especially with large datasets where transferring data between systems is expensive. Example categories of skills that DataChat users leverage are shown in Table 1.

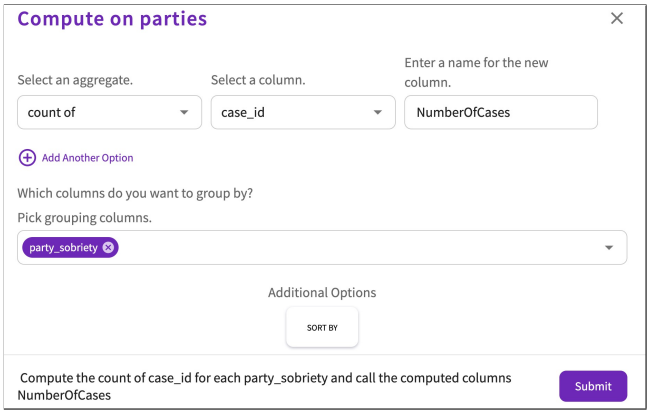
Of particular interest to many users is the ability to augment traditional data preparation steps with machine learning and visualization steps. This seamless transition allows users to iterate on models much faster than with specialized tools. Moreover, by bringing machine learning capabilities such as model training and inference into the data preparation space, the platform allows users the freedom to use more sophisticated techniques. For example, we have seen a number of users adjust from using simple statistical outlier detection methods to ones based on more robust machine learning algorithms.

Table 1: Example DataChat Skills

Data Ingestion	Load data from the file <file name>
Data Exploration	Describe the column <column>
Data Visualization	Visualize <kpi column> using <column>
Data Wrangling	Compute the <aggregate> of <column>
Machine Learning	Train a model to predict <column>

User entry of skill requests takes three main forms shown in Figure 3. The first is via user interface interactions, such as forms, that are converted directly to discrete skill requests. The second is via programmatic APIs that wrap requests to the DataChat platform in more traditional analytics libraries. The third is through directly entering a DataChat GEL sentence in a console-like interface exposed to the user. All three approaches ultimately convert the user's intent to a discrete, parameterized request and only differ insofar as to whether an additional translation step is necessary. For example, the console-like interface requires a natural language parsing step.

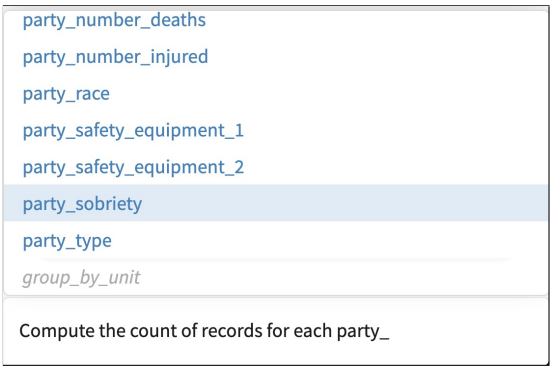
In addition to these three approaches to invoking skills directly, users can also auto-generate a sequence of skills using the natural language to code generation (NL2Code) methods (see Section 4).



(a) A form in the UI.

```
aggregate_data = california_car_collisions.compute(
    aggregate = [Count("case_id")],
    for_each = ["party_sobriety"]
)
```

(b) A Python API call.



(c) Composing a DataChat GEL sentence directly with autocomplete.

Figure 3: The three different methods to enter the same skill.

These approaches offer interaction paradigms that correspond to the current familiarity and comfort levels of users with a wide range of technical expertise.

Many DataChat experts choose to use the console-like interface because it allows fast request entry, but most novice users prefer the visual interface since it provides more guidance on what requests are possible.

For a user of the platform, a skill consists of three parts:

(1) An input specification that is provided by the user.
(2) A parameterized transformation that is defined by the platform.

(3) A set of output artifacts that is received by the user when the skill has finished execution.

2.2 Execution

Skill execution follows a simple pattern. The user first creates a directed acyclic graph (DAG) of skill requests using one of the provided input methods (UI, DataChat GEL, Python API, or the NL2Code methods). Building this DAG does not require executing any computation. The user then takes an action where they would need the results from that set of operations. At this point, the system does the following:

(1) Convert the DAG of skill calls to a set of execution tasks.
(2) Run those execution tasks using compute resources.
(3) Return results to the user based on that execution.

Execution tasks within the framework represent tasks like "Run SQL query" or "Train a machine learning model". In most cases, one execution task encompasses multiple logical skill calls from the user. For performance reasons, the conversion of skill calls to execution tasks is also aware of a caching layer that can execute directly on previous results based on a shared skill sub-DAG.

One major consideration in this approach is delaying the execution of expensive computations until they are actually necessary for the user's experience. This is possible by keeping the logical skill DAG of each request until execution time rather than building subsequent requests on the results of the previous one. An easy example of this optimization is a user projecting columns from a table in a sequence of steps. A naive approach would be to rely on the previous result and simply nest subsequent queries. This would result in deep queries such as:

```
SELECT a FROM (
  SELECT a, b FROM (
    SELECT a, b, c FROM base_table
  )
);
```

In some database engines, this query will incur significant performance costs compared to its flattened equivalent, which is a single query block: `SELECT a FROM base_table;`

By re-evaluating the execution tasks from the complete skill DAG for each request, such optimizations can occur naturally at execution time.

Because the user is responsible for defining the inputs, but not the execution step, the platform can dynamically optimize how a skill is executed. Most execution tasks within DataChat are implemented in both SQL and Python, separately. This approach allows the system to use the appropriate language for a variety of tasks. In most (but not all) cases, SQL is appropriate for data operations. For example, if data is already loaded into memory on a worker for a machine learning task, the platform can run simple data operations there rather than reaching back out to a source database to do so in SQL.

This execution approach allows a range of features to be built using the same logical skills. For example, in a spreadsheet view a user expects to run an operation like filtering values and then immediately see the data result. For large datasets, the application likely wants to simulate this step by providing the user with a sample of the data instead. This behavior can be accomplished naturally by having the application automatically add `Sample/Limit` skills

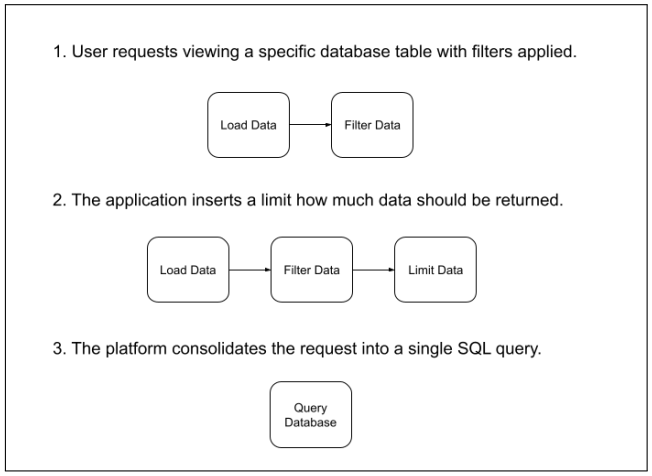


Figure 4: Showing both user intents and application requirements can be optimized in one execution approach.

for the user as shown in Figure 4. To execute correctly, the platform does not need to differentiate between steps the user made and steps that were automatically added. This approach enables the development of sophisticated applications that automate a variety of analysis steps for the user.

2.3 Transparency and Reproducibility

A key aspect of data science work is the production of results such as visualizations, models, or raw data that can be persisted to explain or drive decisions. We refer to these result objects as *artifacts*. Artifacts generally consist of a static representation of the object the user cares about (depending on the type of artifact) as well as instructions for how it was produced. Within DataChat, these instructions are a serialized copy of the skill DAG called a *recipe*. Artifacts can be persisted in a variety of ways, depending on their type, but typically include some metadata (including the recipe) stored in a structured database as well as a reference to the data the artifact represents stored elsewhere.

When saving an artifact, such as a chart, the platform automatically processes the skill DAG in a step called *slicing*. In this step, the system evaluates which steps in the DAG affect the final artifact. All steps that have no effect are removed prior to saving. Additionally, some optimizations typically reserved for execution are applied. For example, some skill calls might be merged if they can be represented by a single skill call. This slicing step produces significantly simpler recipes for individual artifacts created during a longer session of data exploration.

Every artifact is paired with a recipe to allow users to easily view how a result was created. To provide transparency and confidence to casual data users, every skill in DataChat has the ability to explain its behavior to users. For technical users, this is done by providing Python or SQL code that represents the skill. However, this code can often be complex when edge cases and scalability considerations are taken into account. For that reason, the platform also provides a declarative controlled English description of what the skill did. This description of the skill's behavior is based on both the skill

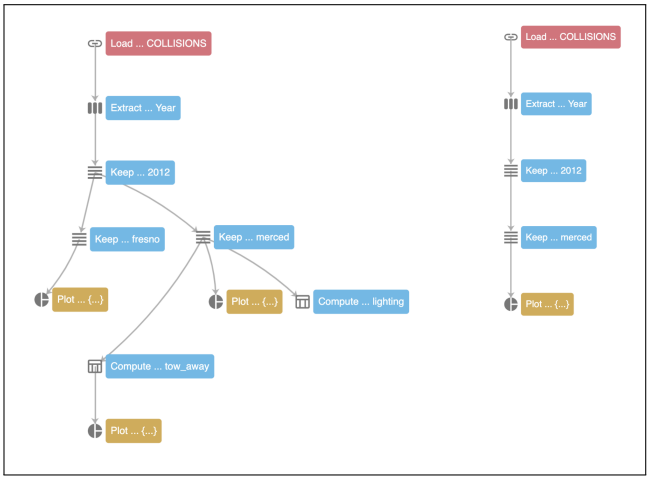


Figure 5: A complex exploratory recipe on the left can be sliced down to a simple linear one automatically.

itself and the inputs the user provided. This descriptive language is referred to as *Guided English Language* or simply *GEL*.

When artifacts are shared between users, the recipe is visible to all users with access. This sharing of recipes allows downstream users to more easily understand the nuances in the data or analysis that is used to generate the artifact. Moreover, the shared recipe allows users to easily replicate the analysis on different data and validate the approach. In many organizations, this type of self-documentation is critical to validate and audit decisions. Having the system automatically create and manage recipes significantly reduces the time users need to spend on documentation.

The other reason for recipes being so prevalent is that they allow updating artifacts on the latest data. For the user, this is as simple as executing the skill DAG again. In most situations, this interaction is presented as a "refresh" because the only difference in the final result will be based on changes in the underlying data. For a more in-depth understanding of how the artifact was produced, a live replay of the steps can be performed, as if an expert was entering the steps for the first time.

Recipes are designed to be edited and updated as requirements change. This way, creators can update steps to keep them aligned with underlying data or process changes. Editing recipes can be done in a variety of ways, the simplest being directly updating the GEL representation of the recipe. Another approach is to step through the recipe one step at a time and manually override any steps that are no longer correct. A more technical approach is to view the skill DAG directly in a graphical form and update parameters, or edges manually.

2.4 Collaboration

Most users of the DataChat platform work in teams. Such teams are frequently multifaceted and are composed of individuals with a range of expertise to solve a variety of problems. Hence, the platform they use must cater to teams working as a unit as well as individuals. Additionally, the platform must be accessible to both technical experts and non-technical experts in various domains.

To facilitate collaboration on active data exploration, the platform is designed to allow multiple users to view and interact with the same data in a user interface. To grant users fine-grained control over data access and sharing, a user's work is organized into individual *sessions*. Sessions consist of one or more underlying skill DAGs, and hence provide logical isolation between multiple analyses that the user may execute. Sessions can be shared with other users to give them access to both the skill DAG as well as data results within that session. Various levels of access privileges can be granted to or revoked from individual collaborators.

Actions taken in a session are tracked in the platform itself rather than the client, so multiple users can maintain a synchronized view of the work. A simple session-level lock prevents concurrent skill requests. This is appropriate for data science workflows because an operation is typically motivated by the immediately preceding request. If two requests are submitted at the same time, and one is run, the other will often not hold the same logical meaning anymore. For example, if one user creates a pivot table, another user's filter operation may no longer make sense. Hence, requests sent concurrently will fail with a message to the user indicating that another execution was already running.

Similar to active collaboration, artifacts saved within the system can also be shared. For most artifacts, sharing directly in the platform is significantly more meaningful than exporting an image or description of the result. For example, plots can be interacted with or even changed to explore data further. Sharing also provides access to the underlying recipe. Permissions for shared artifacts may include taking actions or editing them just like a session. Sharing artifacts is a convenient way for users doing data exploration to get feedback on specific results from subject matter experts.

Sharing with users outside the platform is also an important user flow when working with an entire enterprise organization (because not all employees may be users) or an ad hoc-defined group of recipients. This mode of collaboration is facilitated by allowing artifacts to be associated with a generated secret and key. This secret can be included in requests to authorize access to an artifact rather than a user directly. For some types of artifacts, especially visualizations, it is highly convenient to include this secret in a URL directly to allow sharing a secure link directly to results.

Managing shared objects within the platform is done through a *Home Screen* UI that resembles an operating system file manager. Users can organize artifacts and sessions into folders, which behave both as a container for multiple artifacts as well as an artifact themselves. From the Home Screen, users can share, view, rename, or delete artifacts. Users can also edit the artifact's DAG, or execute the DAG to update the object.

While the Home Screen allows the organization of objects and easy management, it often does not allow users to tell a story with their results. In many enterprise settings, the results of data analysis (charts, tables, and other insights) are presented by copying static images out of data analysis tools into a slide deck. In DataChat, the presentation of results can be done using an Insights Board (IB). An Insights Board is a collection of artifacts presented in a visual layout that the creator can define. An IB is intended to be a presentation of results at the end of an analysis. Functionally, an IB is modeled as a slide/poster in a presentation scenario rather than a traditional operational dashboard. For this reason, IBs allow

arbitrary formatting of artifact positioning and the addition of graphical elements like text boxes. Completely unrelated artifacts can be posted to the same IB, seeing as data science conclusions often depend on multiple different arguments.

A typical user in DataChat will follow the following steps:

(1) Open a session and load in data.
(2) Work in that session by invoking skills.
(3) Share the session to work with coworkers and validate assumptions.
(4) Publish results as artifacts to an Insights Board.
(5) Present an Insights Board with the results of the analysis.

This flow results in a presentation of results that is highly interactive. Because every artifact on the Insights Board is backed by a readable description, questions regarding how results were obtained are trivial to answer. Moreover, because every artifact maintains its own DAG of operations, the presenter (or even the audience) can quickly run additional analyses to delve into the results. This approach allows questions to be answered live when presenting results rather than iterative sessions to reach alignment between stakeholders.

3 SAMPLING AND SNAPSHOTS

This section highlights the built-in features of the DataChat platform that enhance the user experience when working with cloud database systems. The emergence of cloud databases has facilitated enterprise data accessibility to a wider range of users. However, the increasing number of users carrying out complex data science functions using enterprise data also presents a challenge for enterprises – the cost of running analytics on large data volumes stored in cloud databases, and by a larger number of users, can quickly escalate due to the prevalent consumption-based pricing models.

DataChat addresses this challenge through two critical skills integrated into the platform: sampling and snapshots. First, a user can request a sample of the data to create a data analysis recipe, which is typically the initial step in exploring and assessing data for new or ad hoc analyses. DataChat has built-in skills for sampling that enable users to quickly scan even large tables at a fraction of the cost incurred to scan the entire table. This approach generates an intermediate dataset that users can work off to carry out an initial assessment. Sampling rates can be set, and common rates include 10% and 1%. With the sample data, users can visually explore the data in a tabular form, run built-in DataChat data exploration skills to determine characteristics (such as null values), and better understand the data. The system typically uses block-level sampling, resulting in a significantly lower query cost for the initial data exploration.

For instance, a user working with a new IoT dataset had a 6 billion row dataset in a cloud database. By using a 10% sample, they reduced their cloud bill by 10 times because query costs are generally proportional to the size of the dataset being scanned. In the past, the data pipeline had quality issues that were only discovered at the end of the analytics exercise. However, with the sample data, a quick assessment using data summarization and visualization skills in the DataChat platform was carried out to confirm that the data quality was acceptable (e.g., the number of missing values in the sample was within the expected range). This human-in-the-loop approach also provided a better experience for the user by reducing the latency in each step, which is critical to the human experience. The steps of the recipe could then be shared with other stakeholders/collaborators and the sampling step can be removed to generate the final data product in a time- and cost-efficient manner.

The DataChat platform also offers an optimization feature called "snapshots", which are cached copies of tables/queries from a source cloud database. Snapshots are typically the result of a complex and expensive data pipeline. They are often small, less than 100GB, and are stored in a local database instance that has a fixed, low cost, and is not based on consumption-based pricing. Users can develop their recipes using the snapshots, allowing for efficient iteration on the steps that make up the final recipe. Using a snapshot for this type of iterative work provides significant savings as the larger data pipeline does not need to be rerun to verify incremental progress. Snapshots can also be shared among collaborating users who are developing a recipe, providing both cost savings and a common starting point. Snapshots are artifacts and have associated recipes, making it easy to refresh them as needed. Snapshots can also be created from samples of tables in a source cloud database. Thus, snapshots provide an additional abstraction for users to develop their analytics artifacts in a more time- and cost-efficient manner.

The DataChat platform is SOC 2 certified, and can be deployed using a hosted SaaS model or a private SaaS within the customer's virtual private cloud (VPC), making governance and compliance manageable with features like snapshots.

4 NATURAL LANGUAGE INTERFACES

As described in the previous section, the DataChat platform offers a set of data science skills that can be declaratively invoked using their GEL representations. Users can directly invoke these skills on the DataChat platform to create a recipe of skills that solves the analytics problem at hand.

The DataChat platform also provides users with higher-order mechanisms to translate requests in natural language (English) to recipes. Users can use these autogenerated recipes to retrieve the answer to their request or iterate on the recipe to refine the autogenerated steps. In this section, we describe various methods that users can employ for generating analytics recipes (programs) from natural language descriptions in the DataChat platform.

SQL has been the primary subject of extensive research when it comes to the idea of using natural language to create analytics programs. Early Natural Language Interfaces for Databases (NLIDBs) had limited usefulness due to their input grammar, which was tailored to a specific schema. Systems such as Microsoft English [2], Precise [24], Nalix [16], Nalir [15], and Athena [28] have sought to overcome this limitation by relying on semantic models of databases and techniques like dependency parsing to accurately interpret natural language queries.

An alternative to grammar-based methods are neural network-based methods that train sequence-to-sequence neural networks to translate English to SQL [36, 39]. These approaches require a large number of training examples, consisting of the database schema and sometimes even the contents of the entire database during the training phase. Consequently, their application is limited in many

scenarios, especially if the database is large (the training effort can be prohibitively large both from the time and cost perspectives) and the database is changing (may require expensive retraining and methods to create new training examples). An earlier internal version of the DataChat platform tried to pursue this direction building on the initial prototype [14], but found this approach to be quite fragile. High accuracy translation was a challenge and so was the instability associated with "improving" a model with new examples, which would then change the translations generated by an older model, which can surprise existing users.

Recently, the field of NLP has undergone a paradigm shift with the advent of a new class of models known as "transformers" [34]. These models have significantly enhanced the ability of sequence-to-sequence models to learn contextual information just from natural language sentences. The use of transformer models to translate natural language to SQL has been a topic of extensive research, with the various approaches falling into two main categories. The first category involves parameter updates to a large language model (LLM). These methods include TaBERT [40] and TaPas [12], where a transformer model is trained from scratch, and approaches like PICARD [29], SeaD [38], and RatSQL [35] that learn parameters on pre-trained transformer models. The second category involves providing a set of instructions to the language model to generate text (or code) and avoid any direct updates to the underlying model's parameters. This technique is often referred to as "prompt engineering" or "prompting". Here, a prompt is constructed along with the task specification to contain all necessary information about the problem domain to facilitate code generation. The prompt is then fed into the LLM, which in turn generates the code. This area of prompt engineering is evolving fast, and a recent survey paper [17] describes many of the contemporary prompting methods.

Our approach to generating analytics recipes from user queries uses prompt engineering to interact with Large Language Models (LLMs). The prompt engineering approach allows us to easily adapt our solution to new problem domains. From an architectural perspective, it also gives us the flexibility to quickly switch to a different language model or incorporate a mixture of models into our solution.

The prompting approach suffers from the following limitations:

- **Token limit**: LLMs can only process a fixed number of tokens in their context window, which consists of both the input prompt and generated output. As a consequence, LLM prompts need to be precise.
- **Context quality**: The quality of LLM outputs is affected by the quality of the context in the prompt, such as typographical errors, insufficient examples, and ambiguous prompts.
- **Lack of domain knowledge**: LLMs lack specific knowledge about a dataset or a domain, which must be provided through the prompt context.
- **Sensitivity to complexity**: The performance of LLMs degrades as the number of solution steps needed for a task increases. [5].

Our NL-intent-to-code generation system (NL2Code) overcomes these limitations by developing multiple technical components that assist in composing precise prompts. These components combine information from several sources to generate and execute analytics recipes (code) that align with the user's intent.

The key design considerations of our system are:

- **Human-Centric Approach:** LLMs, despite their impressive abilities, are not capable of always replacing human intelligence and intuition. Hence, our analytics code generation system takes a human-centric approach and aims to incorporate feedback from the user whenever feasible. The user has the option to provide feedback, rely solely on the model, or iterate on the analytics recipe with other human experts using the collaboration features of the DataChat platform (cf. Section 2.4).
- **Polyglot Translation:** Our code generation system can produce analysis recipes in various dialects, including SQL, DataChat's Python API, and DataChat GEL (cf. Section 2.1), which has the benefit of being highly interpretable. This flexibility in translation enables our platform to cater to a diverse range of users. Intermediate and advanced SQL or Python programmers have the option to select their preferred target dialect, while novice and non-technical users may prefer to use the more approachable DataChat GEL as their language of choice.
- **Transparency and Interpretability:** Language models are susceptible to hallucinations [42]. Thus the code generation system should not assume the correctness of the generated analytics program. This means that the generated code should be made available to the user. This is in line with our mission to ensure transparency of operations on the DataChat platform as discussed in Section 2.3.

Next, we describe the components of DataChat's analytics code generation system, which is schematically illustrated in Figure 6.

4.1 Code Generator

The code generator in our setup is a GPT-based LLM that has been trained to learn a conditional distribution of the next token given a set of previous tokens. The underlying task that this model accomplishes is that of mapping tokens in the natural language questions to tokens in the target analytics dialect. LLMs are known for their ability to learn from a small number of examples within the prompt context, called "few-shot learning" [3]. These models can perform this task by capturing latent patterns from input-output example pairs without the need to update parameters. LLMs' few-shot and in-context learning abilities allow us to adapt a general-purpose language model to generate analytics recipes. We achieve code generation by providing the model with sample natural language (NL) question-analytics recipe training pairs in the prompt.

It is important to note that our architecture is designed to be extensible, allowing for the current model to be replaced by any custom pre-trained or fine-tuned model in the future. This ease of modification highlights the modular nature of the code generation component in our system.

The choice of dialect to represent analytics recipes in the training dataset is crucial as LLMs need to learn the representation from a small set of examples. Further, LLMs are sensitive to the number of distinct operations that it needs to learn for composing a solution.

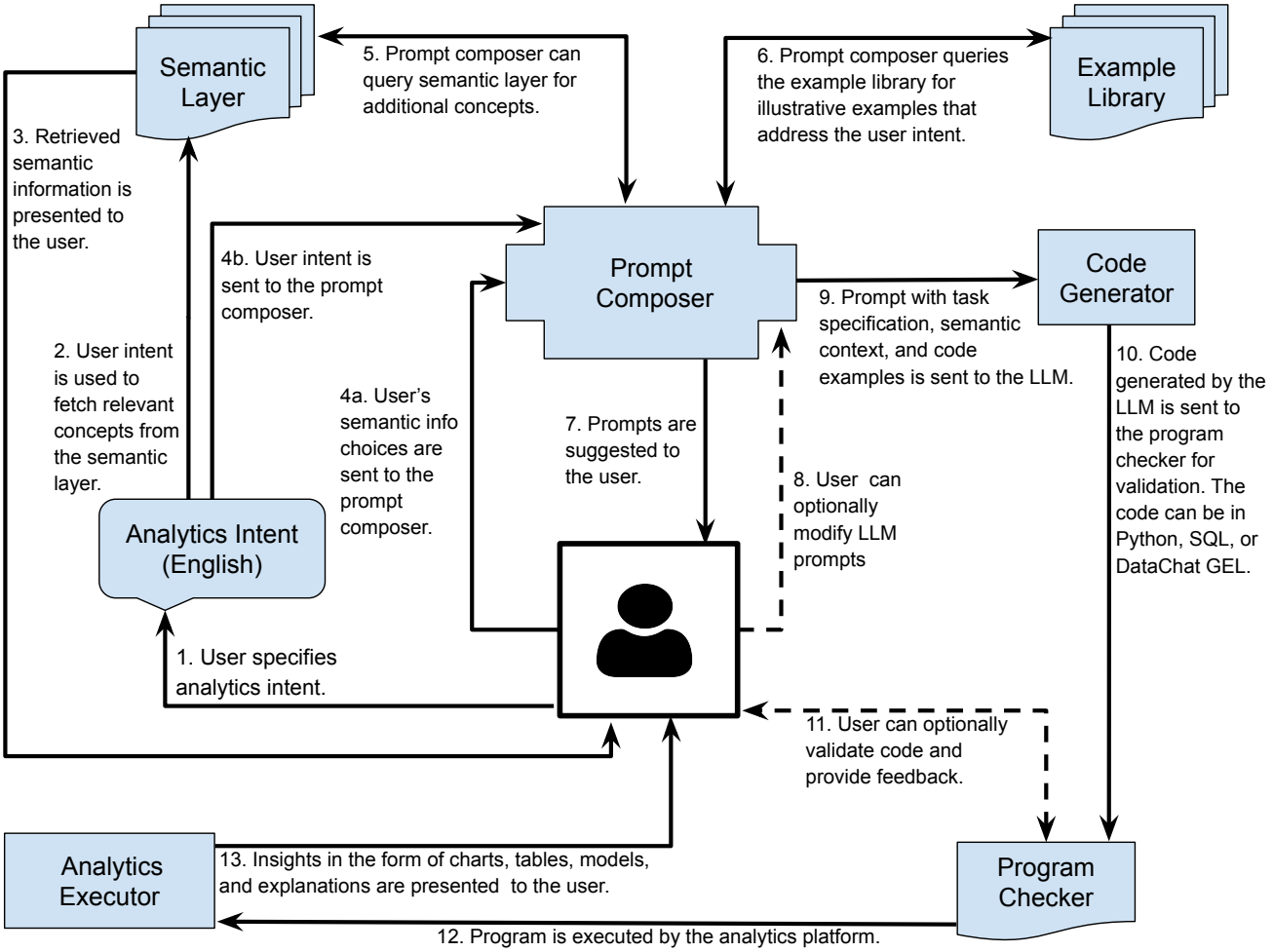


Figure 6: Architecture of DataChat's NL2Code System.

We chose DataChat's Python API as the dialect for representing the analytics recipes.

The DataChat Python API is a wrapper around the skills that are available in the DataChat system. Function invocations in the DatChat Python API have an equivalent DataChat GEL translation. For example, a GEL utterance like "`Compute the Average Age and Median Salary for each JobLevel`" can be represented in the Python API as:

```
data.compute(
    aggregates = [Average('Age')], Median('Salary')],
    for_each = ['JobLevel']
)
```

Our decision to use Python as a dialect is motivated by the fact that the LLM that we use is most proficient in Python. Additionally, the DataChat Python API's skill interface offers a succinct way to invoke data analytics functions, reducing the number of functions that the model has to learn to generate an analytics recipe. Finally, DataChat Python API calls can be easily translated to more DataChat GEL sentences, which makes the programs interpretable and easier to edit.

4.2 Semantic Layer

A semantic layer (SL) is an abstraction that encapsulates domain-specific concepts, links these concepts to the user intent, and offers a contextual representation of these concepts to the LLM. This contextual information provided by the semantic layer enriches the knowledge available to the LLM about the problem domain. The information represented in the SL includes but is not limited to annotations about the data, definitions of domain-specific concepts, metrics, dimensions, and hierarchies.

Let's consider a sales dataset with a column `PurchaseStatus`, that can take the values `Successful`, or `Unsuccessful`. For a question like "How many purchases were successful in the month of April," the model needs to infer that "successful purchases" in the input NL query translates to the predicate `where PurchaseStatus`

= `Successful`. LLMs are not capable of making this association directly from the NL question and the schema alone unless additional information about the problem context is provided to the model. The SL bridges this gap by providing additional context to the LLM for answering the question.

The SL is composed of two main components:

(1) A representation layer that programmatically represents the concepts related to the data schema and problem domain.
(2) A retrieval mechanism that retrieves the concepts that are relevant to the NL query.

The SL first identifies a set of implicit concepts based on attributes derived from the NL query, like keywords, and the problem domain. These identified concepts are then matched with the dataset column names and a known set of concepts in the inferred problem domain. All such matches are weighted based on relevance and the top few are selected for prompt augmentation. The SL outputs need to be as concise as possible to account for the prompt's limited token budget.

4.3 Example Retrieval

In a prompting setting, LLMs are known to exhibit superior performance when the model is provided with a set of examples that are relevant to the task. In the case of analytics program generation, the examples should be representative of the variety of analytics functions and their compositions.

In order to encompass this wide range of analytics functions, we establish a collection of question-solution pairs that vary in both the complexity of the natural language question and the analytics program (see Section 4.7 for problem space characterization). These examples span several problem domains such as sales, finance, and healthcare.

Due to the model's limited token budget, it's crucial to provide examples that are relevant to the user's natural language question. To identify these relevant examples, we first rank the examples in our repository based on their similarity (e.g., cosine) with the user query. Next, from the ranked example list, we select examples that feature a unique set of analytics functions.

4.4 Prompt Composer

The prompt composer serves to integrate information from both the semantic layer and example retrieval components in order to synthesize a prompt that generates valid DataChat Python API code using the LLM. The resulting prompt has the following structure:

(1) API documentation: This section of the prompt contains the names of all the functions in the DataChat Python API, and their signatures.
(2) Examples: The prompt is augmented with sample programs in the DataChat Python API, which are supplied by the example retrieval component. These examples allow for few-shot learning and assist the model in adapting its outputs to the new API syntax. Since the DataChat API is closed-source, it is reasonable to assume that the LLM has no prior knowledge of its syntax. So, we rely on the model's ability to learn within the context of the prompt, with the assistance of a limited number of examples.
(3) Dataset schema and semantic information: We utilize a semantic layer to present the schema of potential datasets and other domain-specific information that is pertinent to the user's natural language (NL) intent to the LLM. Such additional data augments the context information available to the LLM and may be beneficial to code generation.
(4) User intent: Finally, the natural language question by the user is added to the end of the prompt.

The token limits of LLMs introduce a trade-off between the various types of information that can be incorporated into the prompt. The prompt composer can utilize the program space characterization outlined in Section 4.7 to balance the composition of the prompt context. For instance, the prompt composer can decide to omit examples in favor of additional information from the semantic layer in order to address more complex queries.

4.5 Program Checker

The program checker's purpose is to post-process the code generated by the LLM for validating and correcting the program. The program checker converts the LLM-generated analytics program into an abstract representation, keeping track of data and functional dependencies. Using this abstract representation, the program checker performs syntax and type checks and validates the composition of functions in the generated analytics program. Additionally, the checker streamlines the analytics program by removing redundant lines of code such as print statements.

4.6 Human Iteration

The DataChat platform offers users an editor where they can access a post-processed analytics recipe. Users can validate this recipe, execute the code directly, or, with GEL, step through the recipe. By providing this user-friendly editor interface as described in Section 2, the platform enables users to edit, debug, and execute their analytics recipe with ease.

One of the features, which is unique to the DataChat platform, is the ability to debug the generated GEL recipe in an IDE-like manner. Figure 2a shows DataChat's IDE for GEL recipes. The editor has controls for adding breakpoints (indicated by the red dot), executing GEL recipes step-by-step, and pausing the execution at any given point.

The ability to interactively execute and edit the analytics recipe is a powerful paradigm that allows users to validate the generated analytics program on the fly. With this functionality, users can easily and quickly make changes to analytics recipes, allowing them to validate their work without having to go through repeated iterations. Thus, users can save valuable time and execution costs, thereby reducing the overall cost to generate insights in an organization.

The DataChat platform is designed to cater to the needs of a broader range of users by offering a polyglot code generation and execution interface. This means that both novice and expert users can use the analytics recipe generation functionality to quickly find solutions or a good starting point for their analytics problem. Users can then iterate on these recipes to find the final solution that best meets their requirements.

Additionally, as the code generation process involves human input at several stages, the platform can learn and improve over time from human feedback.

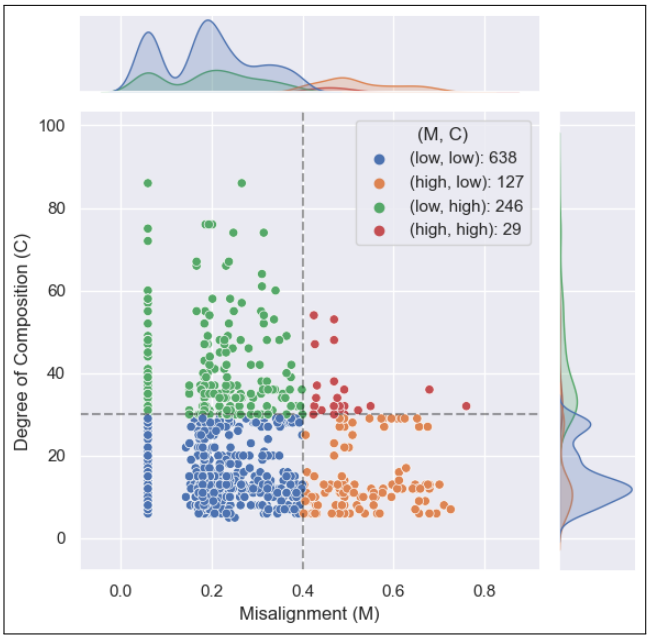


Figure 7: Distribution of all samples from the Spider dev split characterized by misalignment (M) and the degree of composition (C), annotated with the number of points in each zone. Based on the distribution of the scores, the thresholds for M and C were chosen to be 0.4 and 30 respectively. Most samples are characterized as `(low, low)`.

4.7 Experiments

The performance of a text-to-analytics code LLM can vary based on the "difficulty" of a question, which is influenced by factors such as the question structure, the complexity of the solution, and the complexity of the schema [27]. The challenges resulting from these factors necessitate the implementation of distinct solution strategies.

To aid with formulating solutions, we develop a formal characterization of difficulty, which relies on both the question input and the target schema. We devise two metrics, namely:

(1) **Misalignment (M)**: This metric captures the extent to which tokens in the NL query are disassociated with table identifiers and other semantic concepts relevant to the analytics task. This score M is computed as a weighted sum of a query mismatch score s_1 and a schema irrelevance score s_2. s_1 captures the mismatch between keywords from the NL query and the table identifiers, such as column names. s_2 formalizes the difficulty of linking schema identifiers such as column names to real-world concepts.

(2) **Degree of Composition (C)**: This metric measures the functional complexity of an analytics program. For instance, in a SQL query, the functional complexity is a measure of the number of SQL functions that are weighted by their nesting level (a top-level `SELECT` has lesser weight than a `SELECT` in a sub-query) or by their compositional complexity (a `JOIN` operation carries more weight than an aggregation function on a single column).

We evaluate our approach using the development (dev) partition of the Spider text-to-SQL dataset [41]. First, we categorize the dev partition into four groups, jointly classifying the samples into high and low buckets along the dimensions of M and C.

As seen in Figure 7, the Spider dev split follows a long-tailed distribution for both M and C and has very few samples in the high zones. To achieve a more balanced split, we randomly sample an equal number of question-solution pairs from each of the characterized zones to generate the final test set $\mathcal{T}_{spider}$, which is roughly 10% of the entire dev split. Because Spider was publicly available when most pre-trained GPT models were trained, we manually create an additional evaluation set $\mathcal{T}_{custom}$ on tabular data, which was released in the public domain fairly recently.

For each NL query q in the evaluation set $\mathcal{T} = \mathcal{T}_{spider} \cup \mathcal{T}_{custom}$, we generate a DataChat Python API code snippet $\hat{s}$ as the predicted solution. We then calculate the execution accuracy (EA) for each $\hat{s}$ by comparing it against a ground truth solution s. Execution accuracy is a binary (1, 0) metric that compares the results of executing the generated program with a ground truth execution result. The execution accuracy for each (q, s) pair is aggregated into one of the four characterization sectors, as seen in Table 2, We note that performance is the most impacted in the high misalignment and high complexity zone. We also note that higher complexity impacts performance more adversely than higher misalignment.

In summary, our NL2Code system is designed to prioritize user flexibility and program correctness by offering the option of generating code in multiple languages and providing a transparent view of the code being generated. Users can validate their programs in any of the three supported languages (GEL, SQL, and Python) and make any necessary changes before execution. Moreover, users can decide to decompose a complex analytical question into a sequence of easier, targeted questions, whose responses are individually editable. Artifacts generated along the way are also persisted by default, thereby providing the user with a multi-turn program execution paradigm. The easy-to-digest GEL output from the system simplifies the understanding of the analytics code and can be easily translated into other dialects such as Python and SQL.

4.8 Phrase-based Translation

The phrase-based translation method is complementary to LLM-based methods, whereby the input text consists of predefined phrases. These predefined phrases and their corresponding analytics program (or GEL) translations are defined in the semantic layer.

Unlike the LLM-based methods, this method is simple in that extracting information from user utterances is just a lookup of the concepts (phrases) represented in the semantic layer.

The "Visualize" functionality in DataChat drives phrase-based translation. This is part of the extensive repertoire of analytics skills in DataChat as discussed in Section 2.1. The GEL syntax for the Visualize skill is as follows:

```
Visualize <KPI> <grouping phrase> <filter phrase>
```

The KPI entity can be aggregate functions, formulae, or a combination of both on columns in a dataset. The grouping phrase and filter phrase are the equivalents of the group by clause and predicates in SQL. Conjunctions such as "and", and "or" can be used to combine multiple filter phrases. Concepts and their corresponding analytics program definitions can be added to the semantic layer using the `Define` skill.

The phrase-based translation approach is useful when the user's questions have a fixed structure and are driven by terms that can be easily represented as simple English phrases. The advantage of this approach is higher accuracy in translating the intent to the response as the phrases are deterministically matched.

(M, C)	Dataset			
	$\mathcal{T}_{spider}$		$\mathcal{T}_{custom}$	
	samples	**mean EA**	samples	**mean EA**
(low, low)	25	0.84	20	0.65
(low, high)	25	0.76	22	0.59
(high, low)	25	0.80	26	0.73
(high, high)	25	0.68	22	0.25
Mean		**0.77**		**0.57**

Table 2: Mean execution accuracy (EA) on the evaluation sets $\mathcal{T}_{spider}$ and $\mathcal{T}_{custom}$, grouped according to their misalignment (M) and degree of composition (C) scores.

5 CONCLUSIONS

Human efficiency in finding deeper insights hidden in data is currently a critical challenge, as the value/cost of human time in an organization is increasing. This trend is likely to persist into the future. DataChat is addressing this challenge by offering an intuitive data science platform that is increasing the range of users who can perform sophisticated data analytics. The platform is based on a paradigm of a collection of skills that can be expressed via a visual interaction or explicit composition in a controlled-natural language called GEL. It also supports programmatic translation from natural language using an NL2Code component. This component has both a structured phrase-based translation layer, and a more general (unconstrained) LLM-based translation layer to convert natural language (English) to analytics code. The translation layer is polyglot to appeal to a broader class of users. The platform adopts a modular approach across its translation layers as the LLM space is fast evolving and it is important to plan for changes to the specific LLM that is used in the platform. Close integration with a semantic layer is also critical to make the translation layer generate high-quality responses. Finally, the platform includes cost-conscious features such as snapshots and sampling, which are essential in the modern cloud era, especially when the goal is to open analytics to everyone in an organization while managing the operational costs associated with using modern consumption-based cloud database systems.

6 ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under an SBIR Grant No. IIS-1853057. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Mangesh Bendre, Bofan Sun, Ding Zhang, Xinyan Zhou, Kevin Chen-Chuan Chang, and Aditya G. Parameswaran. 2015. DATASPREAD: Unifying Databases and Spreadsheets. *Proc. VLDB Endow.* 8, 12 (2015), 2000–2003. https://doi.org/10.14778/2824032.2824121

[2] Adam Blum. 1999. Microsoft English Query 7.5: Automatic Extraction of Semantics from Relational Databases and OLAP Cubes. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*. Morgan Kaufmann, "", 247–248. http://www.vldb.org/conf/1999/P24.pdf

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). "", Online, 1–25. https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[4] Mihai Budiu, Parikshit Gopalan, Lalith Suresh, Udi Wieder, Han Kruiger, and Marcos K. Aguilera. 2019. Hillview: A trillion-cell spreadsheet for big data. *Proc. VLDB Endow.* 12, 11 (2019), 1442–1457. https://doi.org/10.14778/3342263.3342279

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *CoRR* abs/2107.03374 (2021), 1–35. arXiv:2107.03374 https://arxiv.org/abs/2107.03374

[6] Sisu Data. 2023. Sisu Data: The Decision Intelligence Engine. https://sisudata.com/. Accessed: 2023-02-20.

[7] dbt. 2023. Metrics | dbt Developer Hub - dbt Docs. https://docs.getdbt.com/docs/build/metrics. Accessed: 2023-02-20.

[8] Domo. 2023. The Domo Business Cloud. https://www.domo.com/. Accessed: 2023-02-20.

[9] Encyclopedia Britannica Editors of Encyclopaedia. 2023. Microsoft Excel. https://www.britannica.com/technology/Microsoft-Excel. Accessed: 2023-02-20.

[10] James Gale, Max Seiden, Deepanshu Utkarsh, Jason Frantz, Rob Woollen, and Çagatay Demiralp. 2022. Sigma Workbook: A Spreadsheet for Cloud Data Warehouses. *Proc. VLDB Endow.* 15, 12 (2022), 3670–3673. https://www.vldb.org/pvldb/vol15/p3670-gale.pdf

[11] Google. 2023. Google Collab. https://colab.research.google.com. Accessed: 2023-02-20.

[12] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. TaPas: Weakly Supervised Table Parsing via Pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, Online, 4320–4333. https://doi.org/10.18653/v1/2020.acl-main.398

[13] Hex. 2023. Hex - Do more with data, together. https://hex.tech. Accessed: 2023-02-20.

[14] Rogers Jeffrey Leo John, Navneet Potti, and Jignesh M. Patel. 2017. Ava: From Data to Insights Through Conversations. In *8th Biennial Conference on Innovative Data Systems Research, CIDR 2017, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*. www.cidrdb.org, Online, 1–10. http://cidrdb.org/cidr2017/papers/p87-john-cidr17.pdf

[15] Fei Li and H. V. Jagadish. 2014. NaLIR: an interactive natural language interface for querying relational databases. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu (Eds.). ACM, New York, NY, USA, 709–712. https://doi.org/10.1145/2588555.2594519

[16] Yunyao Li, Huahai Yang, and H. V. Jagadish. 2007. NaLIX: A generic natural language search environment for XML data. *ACM Trans. Database Syst.* 32, 4 (2007), 30. https://doi.org/10.1145/1292609.1292620

[17] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *CoRR* abs/2107.13586 (2021), 1–46. arXiv:2107.13586 https://arxiv.org/abs/2107.13586

[18] Looker. 2023. Looker. https://www.looker.com/. Accessed: 2023-02-20.

[19] R Melville. 1993. Crystal-clear database reporting. *PC World* 11, 5 (1993), 81–81.

[20] Microsoft. 2023. Excel specifications and limits. https://support.microsoft.com/en-us/office/excel-specifications-and-limits-1672b34d-7043-467e-8e27-269d656771c3. Accessed: 2023-02-20.

[21] Microsoft. 2023. Microsoft Excel Spreadsheet Software. https://www.microsoft.com/en-us/microsoft-365/excel. Accessed: 2023-02-20.

[22] Microsoft. 2023. Microsoft Power BI: Data Visualization. http://www.tableau.com. Accessed: 2023-02-20.

[23] Jupyter Notebook. 2023. Jupyter Notebook. https://jupyter.org. Accessed: 2023-02-20.

[24] Ana-Maria Popescu, Oren Etzioni, and Henry A. Kautz. 2003. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces, IUI 2003, Miami, FL, USA, January 12-15, 2003*, David B. Leake, W. Lewis Johnson, and Elisabeth André (Eds.). ACM, New York, NY, USA, 149–157. https://doi.org/10.1145/604045.604070

[25] Qlik. 2023. Qlik: Analytics & Data Integration Platform. https://www.qlik.com/. Accessed: 2023-02-20.

[26] Sajjadur Rahman, Mangesh Bendre, Yuyang Liu, Shichu Zhu, Zhaoyuan Su, Karrie Karahalios, and Aditya G. Parameswaran. 2021. NOAH: Interactive Spreadsheet Exploration with Dynamic Hierarchical Overviews. *Proc. VLDB Endow.* 14, 6 (2021), 970–983. https://doi.org/10.14778/3447689.3447701

[27] Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the Text-to-SQL Capabilities of Large Language Models. arXiv:2204.00498 [cs.CL]

[28] Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R. Mittal, and Fatma Özcan. 2016. ATHENA: An Ontology-Driven System for Natural Language Querying over Relational Data Stores. *Proc. VLDB Endow.* 9, 12 (2016), 1209–1220. https://doi.org/10.14778/2994509.2994536

[29] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. arXiv:2109.05093 [cs.CL]

[30] Tableau Software. 2023. Tableau: The world's leading analytics platform. https://powerbi.microsoft.com/en-us/. Accessed: 2023-02-20.

[31] Chris Stolte, Diane Tang, and Pat Hanrahan. 2002. Polaris: A System for Query, Analysis, and Visualization of Multidimensional Relational Databases. *IEEE Trans. Vis. Comput. Graph.* 8, 1 (2002), 52–65. https://doi.org/10.1109/2945.981851

[32] Chris Stolte, Diane Tang, and Pat Hanrahan. 2002. Query, analysis, and visualization of hierarchically structured data using Polaris. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*. ACM, New York, NY, USA, 112–122. https://doi.org/10.1145/775047.775064

[33] ThoughtSpot. 2023. ThoughtSpot.com - The Modern Analytics Cloud. https://www.thoughtspot.com. Accessed: 2023-02-20.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). "", "", 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[35] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2019. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. arXiv:1911.04942 [cs.CL]

[36] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, "", 7567–7578. https://doi.org/10.18653/v1/2020.acl-main.677

[37] Douglas J Wolf. 1999. *Seagate Crystal Reports 7 for Dummies*. John Wiley & Sons, Inc., Hoboken, NJ, USA.

[38] Kuan Xu, Yongbo Wang, Yongliang Wang, Zihao Wang, Zujie Wen, and Yang Dong. 2022. SeaD: End-to-end Text-to-SQL Generation with Schema-aware Denoising. In *Findings of the Association for Computational Linguistics: NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, Marine Carpuat, Marie-Catherine de Marneffe, and Iván Vladimir Meza Ruíz (Eds.). Association for Computational Linguistics, "", 1845–1853. https://doi.org/10.18653/v1/2022.findings-naacl.141

[39] Xiaojun Xu, Chang Liu, and Dawn Song. 2017. SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning. *CoRR* abs/1711.04436 (2017), 1–13. arXiv:1711.04436 http://arxiv.org/abs/1711.04436

[40] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, "", 8413–8426. https://doi.org/10.18653/v1/2020.acl-main.745

[41] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. *CoRR* abs/1809.08887 (2018), 1–11. arXiv:1809.08887 http://arxiv.org/abs/1809.08887

[42] Chunting Zhou, Graham Neubig, Jiatao Gu, Mona Diab, Paco Guzman, Luke Zettlemoyer, and Marjan Ghazvininejad. 2020. Detecting Hallucinated Content in Conditional Neural Sequence Generation. arXiv:2011.02593 [cs.CL]