SQL Agents: Agentic AI approach for Automated Query Generation and Database Interaction

Het Patel

intern.ai5@adani.com, *patelhetmanojkumar@gmail.com*

Adani – NR: Analytics & AI

Maulik Pandya

Maulik.Pandya@adani.com

Adani – NR: Analytics & AI

Abstract

This paper introduces a robust framework for automated query generation and database interaction leveraging Agentic AI and SQL agents. This framework is designed to generate syntactically accurate queries using a Large Language Model (LLM), allowing user to interact with database in Natural Language (NL). By focusing on query generation, the workflow improves flexibility and control by precisely selecting the database and fetching the schema for the following database for a given input query, ensuring precise data retrieval and interpretation.

I. Introduction

In the current era of data-centric solutions, traditional query execution and database management systems require considerable human expertise. This project proposes a novel automated framework centred around query generation using Agentic AI [1] capabilities. Unlike conventional framework that focus on executing queries, this framework emphasizes generating syntactically valid and contextually relevant SQL queries from natural language inputs. Such an approach provides better auditability, transparency, and adaptability in environments where automated or human-mediated query execution is desired.

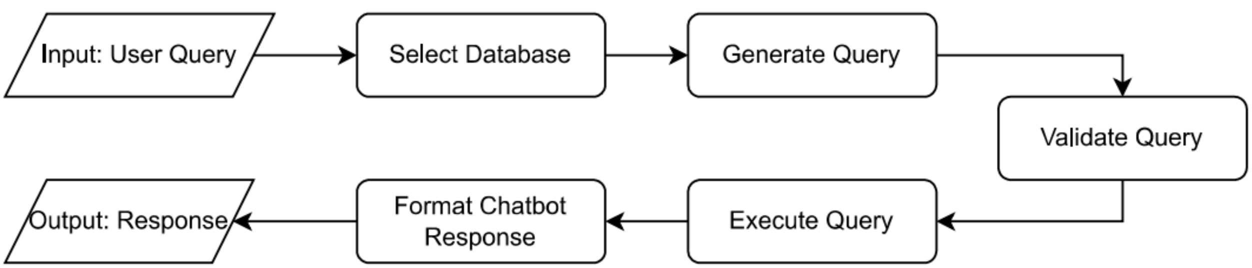

Figure I: Flowchart of the Chatbot Functions

This project involves developing a comprehensive workflow that automates the conversion of various file formats (such as CSV, Excel, Spreadsheets and SQL databases) into SQL database files within a directory. The Graphical Workflow compiled using LangGraph leverages Large Language Models (LLMs) [7] and Langchain – SQLDatabaseToolkit [5] to interpret Input NL Queries, after fetching the schema using SQLDatabaseToolkit – ListSQLDatabaseToolkit & InfoSQLDatabaseToolkit [5] selecting the correct database and table as per Input NL Query using Jaccard Similarity, generate corresponding SQL queries by invoking LLM, validate the SQL Queries using SQLDatabaseToolkit – QuerySQLCheckerTool [5], and execute them on the appropriate database using SQLDatabaseToolkit – QuerySQLDatabaseTool [5]. By integrating functions for schema extraction, selecting database and table, query generation, query validation and result formatting. The Framework of LangGraph [6] compiled flow ensures that the generated SQL queries are both accurate and efficient using LLM and QuerySQLCheckerTool. This approach reduces the need for extensive human intervention and enhances the overall reliability and scalability of database interpretation processes by appending the user query and letting the framework determine the response in the multi-database directory.

II. Functional Concepts

This framework leverages several functional layers that work cohesively to automate the query generation pipeline:

A. **Modular Agent Design:** Each step in the workflow is represented as a dependent function cumulatively for all the previous functions [1, 2, 4], with a clearly defined function and input-output contract. These agents operate as nodes in LangGraph [6], offering a modular and scalable approach if required to deploy the framework on large scale as a System.

B. **Input Format Flexibility:** The framework supports dynamic ingestion of structured files (CSV, Excel, Spreadsheet), converting them into a normalized SQL File, if not in SQL File Format for uniform downstream processing using a Python Library – Pandas [8].

C. **LLM-Driven Reasoning:** The language model used, Llama 3.1 (8B), runs locally using Ollama [7], a platform for serving and interacting with open LLMs. This configuration ensures deterministic, privacy-respecting, and high-performance generation. The LLM interprets user intent, produces valid SQL, and forms contextual responses through engineered prompts.

D. **SQLDatabaseToolkit Integration:** SQLDatabaseToolkit [5] bridges the LLM and the SQL schema, allowing controlled interaction between the generated queries and the underlying database. It acts as a parsing and temporary execution structure that reduces the complexity of SQL construction and aligns outputs with the schema by using its following library-functions:

    a. **ListSQLDatabaseTool:** It stores table names of the selected database in a variable.

    b. **InfoSQLDatabaseTool:** It stores the detailed information about Database, such as schema, metadata and table structures in a variable.

    c. **QuerySQLCheckerTool:** It validates the SQL Queries to ensure whether they are correct, with selected database using LLM.

    d. **QuerySQLDatabaseTool:** It facilitates in executing the validated SQL Query on the selected and connected database for retrieving the results.

E. **State Management with Query Schema:** The query schema maintains the complete state of a query lifecycle, including the natural language input, selected database, extracted schema, generated SQL, and chatbot response for LangGraph Workflow [6]. This enables traceability and debugging and is achieved by using Type Hint (->) punctuation mark.

F. **Semantic Schema Mapping Before query generation**: The schema of the database is semantically analysed. This allows database selection function to understand user query more accurately by fetching to database name, table names, column names, column values and database schema using InfoSQLDatabaseTool and ListSQLDatabaseTool [5,7].

G. **Query Generation, Validation and Execution:** For the selected database's table list, column list, schema and metadata, SQL Query is to be generated by invoking the LLM. Further leading to the validation of the query using QuerySQLCheckerTool and execution of the validated query using QuerySQLDatabaseTool.

H. **LangGraph for Orchestration:** LangGraph [1, 2, 6] facilitates the orchestration of the workflow by treating each function as a graph node with transitions between states. This approach enhances modularity, fault tolerance, and reusability of components.

I. **Formatting the user-friendly response:** Retrieved response from the executed query on the selected database is then converted to user-friendly response by invoking LLM [7].

J. **Human-readable Output Generation Post-query generation:** The framework interprets SQL results to form natural language responses, improving usability for non-technical users by invoking LLM and using Langchain - SQLDatabaseToolkit and LangGraph [5, 6].

III. Framework Architecture

The framework architecture consists of several key components, each playing a vital role in the overall functionality of the framework.

- **Language Model Initialization:** The large language model (LLM) is initialized, setting the model to use Ollama LLM with a temperature of 0 to control the randomness of the output which describes it makes maximum deterministic and focussed outputs resulting in less variation and more predictable outputs. This model is responsible for generating SQL queries from natural language inputs and providing conversational responses based on query results.
- **File Processing and Database Conversion:** A component (code-snippet) of the framework which is present in the function of database selection scans a specified folder for files, converts them into SQLite databases, and stores the connections in a dictionary. It handles .db, .xlsx, .xls, and .csv file formats, ensuring proper encoding such as utf-8 (a character encoding standard used for electronic communication) and table creation. This process involves reading the files, converting them into pandas DataFrames, and then storing these DataFrames in SQLite databases. This framework also handles different encodings to ensure compatibility with various file formats.
- **Schema Information Retrieval:** The framework retrieves schema information for each database, storing it in a dictionary for later use in query generation and validation. This step involves using tools like ListSQLDatabaseTool and InfoSQLDatabaseTool [5] to fetch table names and their respective schemas. This information is crucial for generating accurate SQL queries and validating them against the database structure.
- **LangGraph Query State Schema:** A query state schema is defined using the QueryState [6] class, which includes attributes for the query, selected database, schema, SQL query, validated SQL, result, and chatbot response. This schema ensures that all relevant information is captured and passed through the various stages of the workflow.

Component	**Description**
LLM Initialization	Utilizes Llama 3.1 (8B) with deterministic configuration for predictable outputs.
File Conversion	Transforms .csv, .xls, .xlsx files into .db.
Schema Extraction	Uses tools to list tables and schema for guiding SQL generation. Also captures all query metadata including DB selection, schema, generated SQL, and response.

Component	Description
Compile LangGraph Workflow	Structures the process into modular, reconfigurable stages.

Table I. Framework Components Overview

IV. Framework's Workflow Implementation

The workflow is implemented using LangGraph and consists of several interconnected steps, each designed to handle a specific aspect of the query processing.

- **Database Selection:** The function determines the best database and table based on Jaccard similarity with table and column names. Jaccard similarity [3, 9] J(x,y) is a measure of the similarity between two sets, defined as the size of the intersection divided by the size of the union of the sets. In this context, the function extracts terms from the user's query and compares them with table and column names in the database schemas. The database and table with the highest similarity score are selected for query generation. This ensures that the most relevant data source is chosen for the user's query.

$$J(x, y) = \frac{|x \cap y|}{|x \cup y|}$$

- **SQL Query Generation by invoking LLM:** The function generates an SQL query from the user's natural language query using the LLM [7]. The function considers the selected database schema and ensures that the generated query is syntactically correct and relevant to the user's question.
- **SQL Query Validation:** The function validates the generated SQL query against the database schema to ensure accuracy. This involves checking the query for any invalid columns or syntax errors and using the QuerySQLCheckerTool [5] to validate the query. If any issues are found, the function raises an error, prompting the user to refine their query.
- **SQL Query Execution:** The function executes the validated SQL query and fetches the results. This step involves using the QuerySQLDatabaseTool [5] to run the query against the selected database and retrieve the results. The results are then stored in the query state for further processing.
- **Chatbot Response Generation:** The function generates a user-friendly response based on the query and results using the LLM [7]. This involves formatting the results into a conversational response that is easy for the user to understand. The function ensures that the response is relevant to the user's query and provides meaningful insights based on the query results.

Function	Description
Database Selection	Applies Jaccard similarity between query tokens and schema terms to choose the most relevant database, tables names, column names, column values and database schema.
SQL Generation	Uses LLM to translate natural language into SQL based on selected schema of database.
SQL Validation	Pseudo-validation to ensure logical correctness and schema alignment.
SQL Execution	Execute the generated SQL Query in the selected database table.
Response Creation	Converts SQL output into a human-readable response format.

Table II. Query Generation Workflow

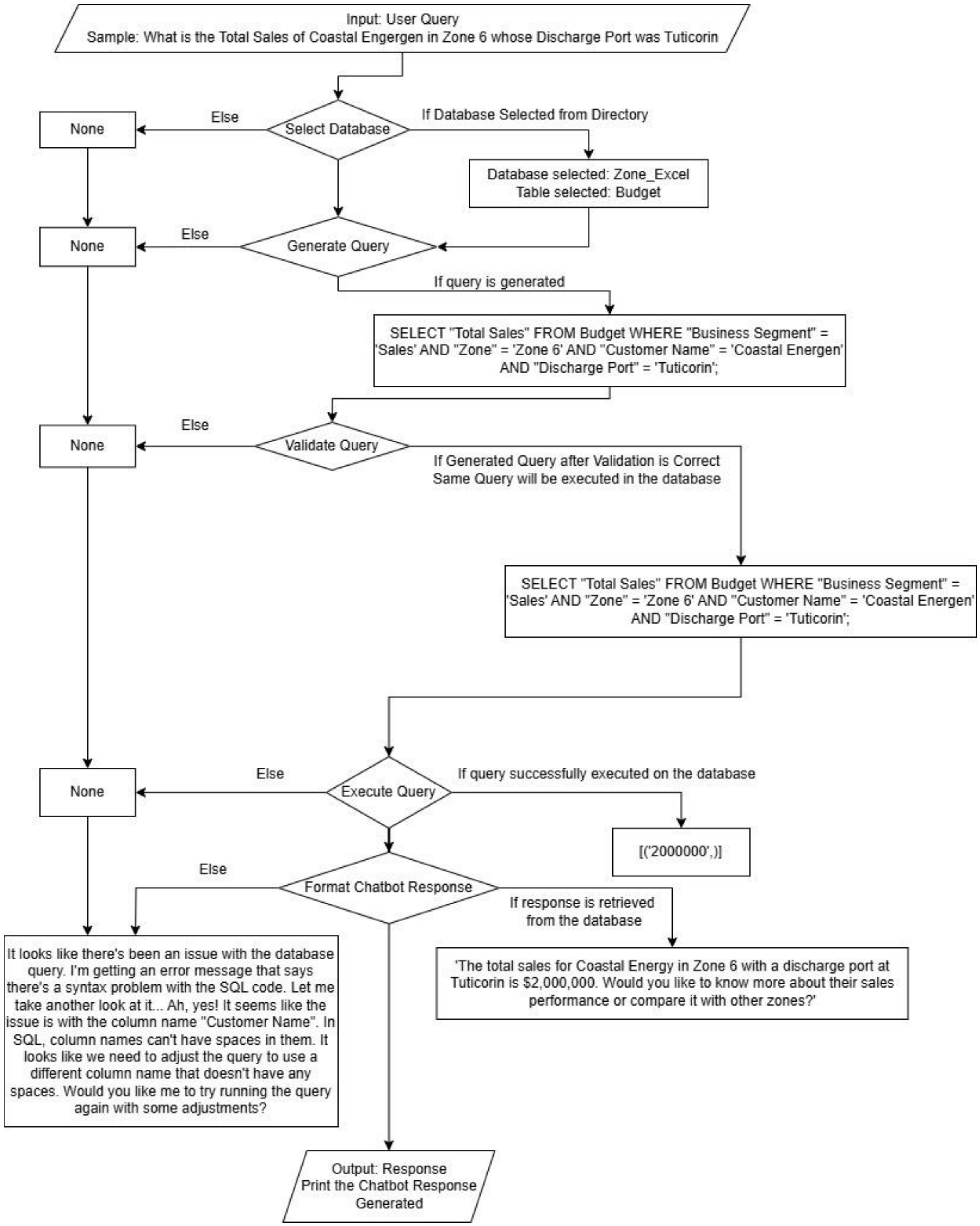

Figure II. Query-Response Flowchart

V. Experiment Setting:

1. **Environment:** The experiments were conducted on a system named with an 11th Gen Intel Core i5-1145G7 processor running at 2.60GHz 1.50GHz, 16GB of RAM, and Windows 11 Enterprise (version 22H2). The system had a 64-bit operating system and no pen or touch input. Ollama Llama3.1:8b is used as LLM for this workflow. Time taken to generate response from a given user is about 3 to 5 minutes depending on the size of the database file.

2. **Datasets:**

Datasets	**Description**
atmspl_trip_details	This dataset tracks coal quantities transported from mining sites to thermal power stations via rake, offering real-time insights for optimizing logistics and enhancing energy production efficiency.
Customer Portfolio	This dataset includes customer-specific details, industry types, plant locations, coal types, and quantities, providing comprehensive insights for optimizing coal transport and energy production.
diabetes	It is the publicly available dataset Glucose level, Pregnancy Results etc. of a person with diabetes with diabetic results which are used by students for academic studies.
IFDP	This dataset includes forecasted and actual coal prices on time-zone, primarily used in coal trading. It provides crucial insights into market analysis and strategic decision-making in the coal industry.
Meeting Updates	This dataset contains the description of meetings conducted with Customers and the appropriate or strategic actions decided to be taken.
road	It is a publicly available Road Accident dataset for academic practice.
Stock Ageing	This dataset identifies the time periods during which specific coal grades are present at various ports, providing essential information for inventory management and logistical planning in the coal industry.
tps_trip_details	This dataset details the number and positions of rakes dispatched from PEKB to four different thermal power stations, providing crucial insights for optimizing coal transport logistics and power station operations.
Titanic	It is the publicly available Titanic Survived and Died people's dataset.
Zone Excel	This dataset includes tables namely Updates and Issues, Meeting Updates, Budget Details, Input Data, and Dispatch Market Share. It basically provides zone-wise inventory data, including sales and dispatch figures, offering valuable insights.

Table III. Datasets and its Description

VI. Results and Discussion

The framework demonstrates considerable advancements in automating query generation. Key takeaways include:

- **Efficiency:** Rapid transformation from natural language to SQL query, reducing dependence on data analysts.
- **Scalability:** Handles multiple databases and formats with minimal manual setup.
- **Flexibility:** Adaptable to various enterprise environments, particularly due to its focus on generation, not execution.
- **Accessibility:** Enables non-technical users to access structured data using simple queries.

User Query	Validated SQL & Response	Chatbot Response	Remarks
How many transporter are ATMSL?	SELECT COUNT(sn) FROM atmspl_trip_details WHERE transporter = 'ATMSL';	The number of transporters that are ATMSL is 4. Would you like to know more about what ATMSL means or how these transporters operate?	Correct Response Generated.
Give me all the Customer Name of Zone 5	'SELECT "Customer name" FROM Customer_Portfolio WHERE Zone = 'Zone 5';	Here's a natural conversational response: Okay, I've got the list of Customer Names for Zone 5. It looks like we have a total of 59 customers in this zone. Let me break it down for you:We have companies like Adani Wilmar, Ambey Metallics, BMM Ispat, and many more. There are also some well-known brands like Grasim Harihar, JSW Cements Ltd, and The India Cements Ltd.If you'd like to get a specific customer's information or filter the list by any criteria, just let me know! I'm here to help.	It should have generated all the 59 Names even in the chatbot response. As it has retrieved after SQL Query Execution.
On 15 Nov 2024, What were the Action when Rajeev Dalmia Visited	SELECT "Action" FROM Meeting_Updates WHERE "Meeting Date" = '15-Nov-24' AND "Name of the person visited" = 'Rajeev Dalmia';	It looks like you're looking for information about Rajeev Dalmia's visit on November 15, 2024. According to the notes, when he visited, it was expected that operations would be up and running by March 2025, which would allow for cargo transportation from Haldia/Dhamra. Would you like me to clarify anything else about this?	Correct Response Generated.
What is the Total Sales of Coastal Engergen in Zone 6 whose Discharge Port was Tuticorin	SELECT "Total Sales" FROM Budget WHERE "Business Segment" = 'Sales' AND "Zone" = 'Zone 6' AND "Customer Name" = 'Coastal Energen' AND "Discharge Port" = 'Tuticorin';	The total sales for Coastal Energy in Zone 6 with a discharge port at Tuticorin is $2,000,000. Would you like to know more about their sales performance or compare it with other zones?	It has generated answer in $ instead of than in ₹ or Rs.

User Query	Validated SQL & Response	Chatbot Response	Remarks
What is the average aging of all stocks of Zone 1	SELECT AVG(T1.Aging) FROM Stock_Ageing AS T1 WHERE T1.Zone = 'Zone 1'	The average age of all stock in Zone 1 is approximately 133 years! That's quite impressive (or perhaps concerning?) for a zone that's likely home to many long-lived and possibly iconic stocks. Would you like me to break down the data further or compare it with other zones?	Failed to decode Ambiguity, as the answer is in days and not in years.

Table III. Query-Response Table

VII. Limitations and Challenges:

- **LLM Misinterpretation due to failure in Disambiguation [9]:** Occasional misclassification of database context, especially in large or complex schemas. Difficulty in processing databases with >40 columns or >3000 rows due to token limits as per the current CPU System Requirements as mentioned. Examples are: atmspl_trip_details.csv, tps_trip_details.csv, Zone Excel.xlsx with large number of datapoints, columns and sheets.
- **Initial Model Limitations:** Earlier models like SQLCoder-7B required extensive fine-tuning and were inefficient for dynamic data in the enterprises.
- **Scalability Constraints:** Processing speed may decrease with increasing number of tables and nested queries.
- **Transition to Local Environment:** Shifted from cloud-based models (ChatGroq on Colab) to local deployment using Ollama Llama 3.1 to ensure data privacy and security.
- **Limitations of Existing Tools:** Tools like create_sql_agent worked well for public datasets but underperformed on enterprise data. The adoption of SQLDatabaseToolkit improved control and output predictability as described.
- **Evaluation of Matching Techniques:** Initial experiments with Minimum Edit Distance which measures minimum edits to transform one string into another, Weighted Edit Distance which assigns weights to various edit operations and Porter Stemming Algorithms removes common suffixes to normalize word forms, which was replaced with Jaccard Similarity, which showed superior performance in mapping queries to relevant database structures [3, 9].

VIII. Conclusion

The developed Enterprise Chatbot framework automates SQL query generation from natural language inputs, reducing the need for Database Expert and enabling non-technical users to query structured data effectively. It supports multiple databases and formats, ensuring data privacy through local deployment and enhancing accuracy with Jaccard Similarity for database selection. However, it faces limitations such as occasional misclassification in large or complex schemas, difficulty processing extensive databases due to token limits, and decreased processing speed with complex queries. Initial models required significant fine-tuning. Despite these challenges, the framework demonstrates significant achievements in automating query generation, handling diverse data formats, and providing predictable outputs.

IX. References

1. Chandra Maddila, Negar Ghorbani, Kosay Jabre, Vijayaraghavan Murali, Edwin Kim, Parth Thakkar, Nikolay Pavlovich Laptev, Olivia Harman, Diana Hsu, Rui Abreu, and Peter C. Rigby. "AI-Assisted SQL Authoring at Industry Scale." arXiv preprint arXiv:2407.13280 (2024).
2. Qinggang Zhang, Shengyuan Chen, Yuanchen Bei, Zheng Yuan, Huachi Zhou, Zijin Hong, Junnan Dong, Hao Chen, Yi Chang, and Xiao Huang. "A Survey of Graph Retrieval-Augmented Generation for Customized Large Language Models." arXiv preprint arXiv:2501.13958 (2025).
3. Vikas Thada and Vivek Jaglan. "Comparison of Jaccard, Dice, Cosine Similarity Coefficient to Find Best Fitness Value for Web Retrieved Documents Using Genetic Algorithm." International Journal of Innovations in Engineering and Technology, vol. 2, no. 4, pp. 202-205, 2013.
4. Chen Shen, Jin Wang, Sajjadur Rahman, and Eser Kandogan. 2024. Demonstration of a Multi-agent Framework for Text to SQL Applications with Large Language Models. In Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM '24). Association for Computing Machinery, New York, NY, USA, 5280–5283.
5. Langchain - SQLDatabaseToolkit
6. Langchain - LangGraph
7. Ollama
8. Pandas
9. NPTEL IIT Kharagpur – Natural Language Processing by Pawan Goyal