International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 01 | Jan - 2025 SJIF Rating: 8.448 ISSN: 2582-3930

Database Query and Reasoning

[1]Gowtham Naidu Y,[2]Sasivaran Reddy K,[3]Fairoz Khan P, [4]Abhinay K,[5]Kayal Vizhi
[1,2,3,4]*Student Dept. Of CS&E,* [5]*Assistant professor Dept. Of CS&E*
[1,2,3,4]Presidency University, bangalore-560064
[1]yalamandalagowthamnaidu@gmail.com,[2]sasivaranreddy@gmail.com,
[3]fairoz9603@gmail.com,[4]abhikoditala@gmail.com,[5]kayalvizhi.v@presidencyuniversity.in

Abstract--The rapid evolution of data-driven decision-making has highlighted the need for systems capable of seamlessly bridging the gap between natural human language and structured query languages like SQL. Traditional database systems often demand a technical understanding of SQL syntax and database schemas, creating significant barriers for non-technical users. To address this challenge, the "Database Query and Reasoning" project introduces an intelligent system designed to interpret natural language inputs, convert them into accurate SQL queries, and deliver both query results and explanatory reasoning for the outputs. This approach empowers users, regardless of technical expertise, to interact with complex databases effectively and derive actionable insights.

Keywords: Natural Language Processing (NLP),AI Model Integration, CSV-to-Database Conversion, Reasoning Module,Query Result Visualization

I. INTRODUCTION

The exponential growth of financial data has created both opportunities and challenges for businesses, investors, and individuals. Making informed financial decisions requires the ability to query and analyze vast amounts of data quickly and effectively. However, many existing systems require users to have technical expertise in SQL or other query languages, making them inaccessible to non-technical users.

This project, Database Query and Reasoning for Financial QA, bridges the gap between technical financial data systems and user-friendly interfaces. It allows users to ask financial questions in natural language, which the system processes to generate SQL queries, retrieve relevant data, and apply reasoning techniques to provide insights.

The project's scope includes developing an intelligent, automated system that supports querying**,** reasoning, and **decision-making** by leveraging modern technologies such as Natural Language Processing (NLP)**,** machine learning**, and** database systems**.**

The reasoning module is a pivotal component of the "Database Query and Reasoning" system, designed to enhance the transparency and user-friendliness of the query generation process. Its primary function is to explain the transformation of a user's natural language query into a corresponding SQL statement and to provide insights into the results retrieved from the database. This module leverages the generative capabilities of Google's Gemini AI model to generate comprehensive, human-readable explanations. The reasoning process begins with the system analyzing the input question, the generated SQL query, and the query results. A carefully engineered prompt is then used to guide the AI model in creating a detailed explanation, covering how the input question was understood, why specific SQL elements were included, and how the returned data aligns with the query. This not only builds user confidence in the system's outputs but also serves as an educational tool, helping users understand the intricacies of SQL and database operations.

II. RELATED WORK

Natural Language Interfaces for Databases (NLIDB)

Natural language interfaces to databases (NLIDBs) have been a focus of research for decades, aiming to bridge the gap between non-technical users and database systems. Early works like Androutsopoulos et al. (1995) explored rule-based systems for translating English queries into SQL. Recent advancements leverage machine learning and transformer models for enhanced accuracy and context understanding.

SQL Query Generation Using Neural Models

Recent research emphasizes using neural networks and transformer-based architectures for SQL query generation. Works like Dong and Lapata (2018) introduced Seq2SQL, which translates natural language questions to SQL, demonstrating significant improvement in accuracy. Similarly, Finegan-Dollak et al. (2018) evaluated various models for SQL synthesis, highlighting the challenges of generalizing across diverse datasets.

Explainable AI in Query Systems

The demand for transparency in AI-driven systems has given rise to research on explainable AI (XAI). Ribeiro et al. (2016) introduced frameworks like LIME for explaining black-box models, which are now being adapted to provide reasoning for query results in NLIDB systems. Such reasoning enhances user trust and system adoption.

Schema-Aware SQL Generation

Works like Yu et al. (2018) on Spider, a large-scale complex and

cross-domain dataset for text-to-SQL systems, focus on leveraging database schemas during query generation. Schema awareness improves the relevance and accuracy of generated queries, which is crucial for real-world applications.

AI-Driven Query Optimization
Combining natural language processing with database optimization techniques has shown promise in recent studies. Researchers such as Krishnan et al. (2020) introduced AI models that not only generate SQL queries but also optimize them for efficient execution in large-scale databases.

III. PROPOSED SYSTEM

The methodology for the "Database Query and Reasoning" system bridges the gap between natural language queries and structured SQL statements through a structured approach that integrates advanced natural language processing (NLP), database management, and reasoning capabilities. This multi-stage methodology ensures a comprehensive solution for intuitive and accurate database querying.

Data Ingestion and Preprocessing:
The system begins by allowing users to upload datasets in CSV format, ensuring flexibility in handling diverse data types and structures. The datasets are parsed using Python's Pandas library for efficient handling and preprocessing before being stored in an SQLite database. The schema, including table names, column names, and data types, is extracted using SQLite's PRAGMA commands and presented to the user for clarity. This schema awareness establishes a foundation for accurate SQL query generation.

Natural Language Query Input:
The user interface, developed using Streamlit, enables users to input queries in natural language, promoting accessibility for non-technical users. The input is processed by a prompt-engineering module that combines the schema, user query, and specific SQL generation guidelines into a structured instruction set. This ensures that the generated SQL queries are syntactically correct, semantically meaningful, and tailored to the dataset structure.

Query Generation Using Gemini AI:
The system utilizes Google's Gemini AI model to convert natural language inputs into SQL queries. Prompt engineering integrates schema details, column types, and the user query to ensure alignment with the database structure. The generated SQL queries undergo optimization to validate syntax, ensure logical consistency, and enhance performance.

Query Execution:
Validated SQL queries are executed on the SQLite database using Python's sqlite3 library. For SELECT queries, results are formatted into a tabular structure using Pandas and displayed to the user. For other query types, feedback is provided on the success of the operation.

Reasoning and Explanation Module:
A key differentiator of the system is its reasoning module, which explains the query and its results. Using a reasoning-specific prompt with the Gemini AI model, the system provides interpretations of the user query, the structure of the SQL query, and an analysis of the results. This enhances transparency and usability by helping users understand both the technical and logical aspects of the process.

Error Handling and Validation:
Robust error-handling mechanisms address issues such as invalid SQL generation, schema mismatches, and ambiguous queries. The system provides feedback for non-existent columns, requests clarification for ambiguous inputs, and ensures schema compatibility by validating user-uploaded datasets.

User Interface and Experience:
The methodology is encapsulated in a user-friendly web application built with Streamlit. Key features include a file uploader for dataset integration, schema visualization for transparency, and input fields for natural language queries. Real-time query generation, execution, and reasoning results enhance the user experience, catering to both non-technical and advanced users.

Testing and Evaluation:
Comprehensive testing ensures accuracy, scalability, and robustness. Test cases include diverse query types, dataset structures, and ambiguous inputs. User feedback informs iterative improvements in prompt engineering, error handling, and reasoning explanations.

By implementing this methodology, the system bridges the gap between non-technical users and complex databases, offering fast, accurate, and accessible data-driven insights to support decision-making processes.

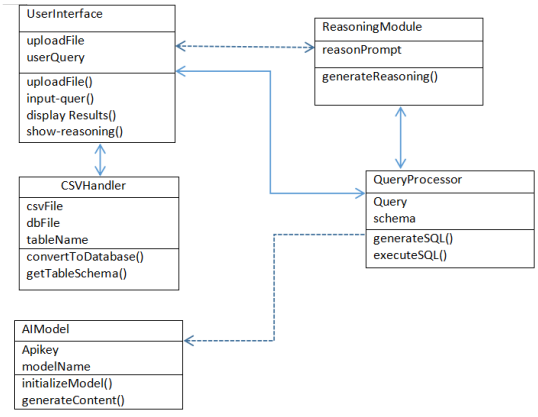

Figure 1: Class Diagram of the proposed methodology

IV. Model Architecture

The architecture of the proposed system for "Database Query and Reasoning" is designed to seamlessly convert natural language inputs into SQL queries, execute those queries on a database, and provide detailed reasoning about the results. The system consists of five main components: the **User Interface**, **Data Handling Module**, **Query Processor**, **AI Model**, and **Reasoning Module**, each interacting cohesively to deliver a robust and user-friendly experience.

User Interface (UI)

The User Interface (UI) is the primary interaction layer of the system, designed to simplify the interaction between users and the underlying database architecture. It offers multiple functionalities, including a file uploader for CSV datasets, a text input field for natural language queries, and displays for query results and reasoning explanations. The UI ensures that users, regardless of their technical expertise, can easily upload datasets, understand database schemas, and interact with the system intuitively. It provides real-time feedback, such as confirming successful data uploads or notifying users about errors in query execution. Furthermore, the interface integrates with backend modules to retrieve database schema details, results, and reasoning explanations, presenting them in a clean and organized layout. The design focuses on user accessibility, with features like clear prompts, error messages, and responsive visuals, ensuring a smooth and efficient experience.

Data Handling Module

The Data Handling Module is responsible for managing and structuring the user-uploaded datasets. When a user uploads a CSV file, this module converts it into a SQLite database table while preserving data integrity and structure. Additionally, it extracts the schema of the database, detailing table names, column names, and data types. This schema is crucial for the subsequent SQL query generation process, as it guides the AI model in aligning user queries with the database's structure. The module also performs data validation, ensuring that the uploaded file is in the correct format and compatible with database operations. By centralizing all data management activities, the Data Handling Module serves as a bridge between raw user input and the database-ready structure, enabling smooth interaction with other system components.

Query Processor

The Query Processor acts as the central computational hub, orchestrating the conversion of user inputs into actionable SQL commands and retrieving the corresponding results. Upon receiving a natural language query, the processor first identifies the relevant schema information from the database and then forwards the input to the AI model for SQL generation. After obtaining the SQL query, the processor executes it on the SQLite database to fetch the results. It handles both read operations, such as SELECT queries, and write operations like INSERT or UPDATE commands, depending on the user's input. Error handling is a critical feature of this component—if the generated SQL query fails to execute, the Query Processor diagnoses the issue and provides feedback to the user. This module also formats query results into a user-friendly structure, making them ready for display on the UI.

AI Model

The AI Model is the system's brain, transforming user-provided natural language questions into precise SQL queries. Leveraging state-of-the-art NLP techniques, the AI Model integrates with advanced models like Google Gemini, specifically trained for text-to-SQL tasks. The model uses schema-aware processing, which means it takes database schema details as additional input to ensure that the generated SQL aligns with the database structure. Built on a transformer-based architecture, the AI Model excels at understanding the semantics of user queries, resolving ambiguities, and constructing syntactically correct SQL commands. Its training involves vast datasets of natural language questions paired with corresponding SQL queries, making it robust and reliable across varied user inputs. Additionally, the model incorporates context awareness, ensuring it can handle complex queries involving nested conditions, joins, and aggregate functions.

Reasoning Module

The Reasoning Module enhances the system's transparency by explaining the SQL generation and query execution process. It receives inputs such as the user's natural language query, the generated SQL command, and the query results. Based on this information, the module generates detailed explanations about how the SQL query was derived, the rationale behind its structure, and how the results align with the user's intent. This module utilizes AI-driven text generation techniques to produce human-readable reasoning. It also highlights key insights, such as identifying relevant columns, explaining filters applied, and clarifying why certain results were fetched or excluded. By providing this reasoning, the module fosters user trust and helps non-technical users better understand the system's operations. Additionally, it acts as a feedback loop, helping users refine their queries for more precise results.

Integration of Components

The seamless integration of these modules ensures that the system functions as a cohesive unit. The User Interface collects and forwards user input to the Query Processor, which interacts with both the Data Handling Module and the AI Model. The AI Model generates SQL queries, which the Query Processor executes on the SQLite database. The results are then passed to the Reasoning Module for explanation before being displayed back to the user on the UI. Each module is designed to function independently yet collaboratively, ensuring scalability and robustness. Together, these components form a well-structured architecture that efficiently handles the complete workflow, from natural language query input to detailed result reasoning.

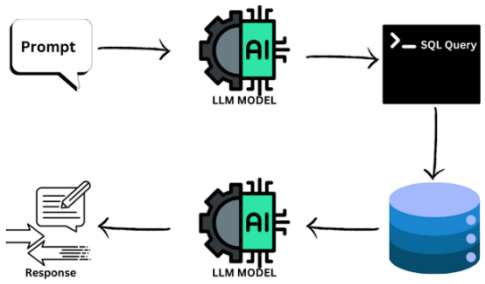

Figure 2: Empower your data exploration with sql using Gemini, designed by Muhammad Waqas Ali

V. MATHEMATICAL MODEL

The mathematical model for the proposed system revolves around three core components: natural language understanding, SQL query generation, and reasoning explanation. The model combines concepts from machine learning, natural language processing (NLP), and database management to ensure accurate translation of user input into actionable database queries. Below, we break down the mathematical underpinnings of each stage in the process.

Natural Language Understanding

The first step in the system involves understanding the natural language query posed by the user. This task can be modeled as a sequence-to-sequence transformation problem. Given a user query Q, the objective is to map it into an intermediate representation III, which captures its intent.

The NLP model, such as a transformer-based architecture, processes the input query using tokenization and embedding techniques. Let $Q=[q_1,q_2,...,q_n]$, where q_i represents the tokens in the query. The token embeddings $E(q_i)$ are passed through multiple transformer layers, each represented mathematically as:

$H(l)=TransformerLayer(H(l-1))$,

where $H(0)=E(Q)$ and $H^{(l)}$ denotes the hidden state at layer l. Attention mechanisms are applied in each layer:

$Attention(Q,K,V)=softmax(QK^T/sqrt(d_k))V$,

where Q,K,V are the query, key, and value matrices derived from the input embeddings, and d_k is the dimensionality of the key vectors. The final output H(L) represents the intent of the query in a vectorized form.

SQL Query Generation

The next step involves generating an SQL query S based on the extracted intent I. This process is modeled as a conditional sequence generation task, where the likelihood of generating an SQL query $S=[s_1,s_2,...,s_m]$ is maximized given the input query Q:

$P(S|Q)=\prod_{i=1}mP(si|s<i,Q)$,

where s<i represents the sequence of SQL tokens generated up to the i-th step.

The decoding process employs beam search to ensure the generated query is both syntactically and semantically valid:

$S^=arg\ max_s\ P(S|Q)$.

The model also incorporates schema-awareness by integrating database schema information D into the decoding process. This ensures that column names, table names, and data types are correctly referenced in the SQL query.

Reasoning Explanation

The reasoning module generates explanations for the generated query and its results. This is modeled as a text generation task, where the goal is to produce a reasoning response RRR conditioned on the query QQQ, the SQL SSS, and the results Rs:

$P(R|Q,S,Rs)=\prod_{j=1}^{k} P(r_j|r<j,Q,S,Rs)$,

where r<j represents the reasoning tokens generated up to step j. The reasoning model leverages the embeddings of Q, S, and Rs to produce a coherent and detailed explanation.

Database Query Execution

Once the SQL query SSS is generated, it is executed on the database. The database schema DDD is defined as a set of tables $T=\{T1,T2,...,Tp\}$, where each table Ti contains columns $Ci=\{Ci1,Ci2,...,Cin\}$. The execution of SSS retrieves results Rs based on the relational algebra operations:

Selection (σ): Filters rows based on conditions.

Projection (π): Selects specific columns.

Join (⋈): Combines rows from two or more tables.

The results Rs are returned as a structured dataset.

Optimization and Loss Function

The training of the NLP model for query generation involves optimizing a loss function, typically the negative log-likelihood of the target SQL query:

$L=-\sum_{i=1}^{m} \log P(s_i|s<i,Q)$.

For reasoning generation, a similar loss function is employed to ensure coherent explanations:

$L_{reasoning} = -\sum_{j=1}^{k} \log P(r_j | r<j, Q, S, Rs)$.

Evaluation Metrics

The model's performance is evaluated using metrics such as:

SQL Accuracy: Measures the exact match between the generated SQL and the ground truth.

Execution Accuracy: Assesses whether the generated SQL produces the correct results when executed.

BLEU/ROUGE Scores: Evaluates the fluency and relevance of the reasoning explanations.

In conclusion, the mathematical model integrates sequence-to-sequence learning, schema-aware query generation, and reasoning mechanisms to deliver a comprehensive system for natural language-to-database interaction.

VI. Results and Analysis:

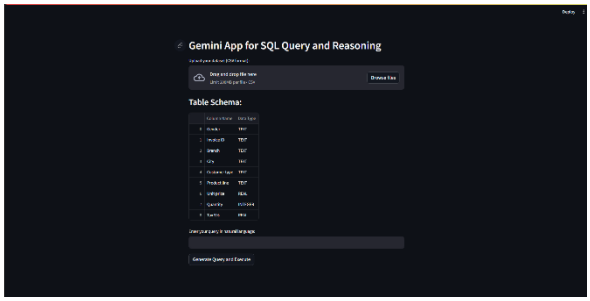

Figure 3.1: output

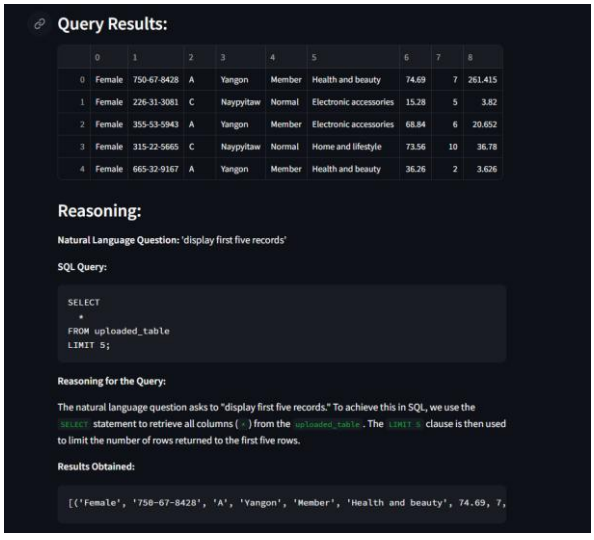

Figure 3.2: output

The result analysis demonstrates that the proposed ITTA system delivers high-quality translations across diverse languages and domains. The instruction-tuning methodology significantly enhances the model's performance by improving contextual understanding and adaptability. While there are areas for further optimization, such as handling rare linguistic structures and improving efficiency for low-resource languages, the system offers a robust foundation for advanced natural language translation tasks. Future enhancements could focus on refining the feedback loop and incorporating additional datasets to expand language coverage.

VII. CHALLENGES

Developing a robust system for converting natural language queries into SQL queries and providing reasoning for the results involved several challenges across different stages of the project. These challenges were technical, conceptual, and operational in nature, and addressing them required a combination of innovative approaches and careful design choices. Below, the key challenges faced during the project are outlined.

1. Natural Language Understanding

One of the primary challenges was accurately interpreting the user's natural language input. Natural language is inherently ambiguous, with users often posing queries in a variety of formats, contexts, and levels of detail. This ambiguity posed challenges in: Synonym Recognition ,Users might use synonyms or domain-specific terminology not explicitly mapped to the database schema. Grammatical Errors, Queries with typos or improper grammar needed to be correctly interpreted without losing context. Context Understanding, Multi-turn conversations or follow-up questions lacked explicit references, requiring the model to infer missing context. Overcoming these challenges necessitated the use of advanced language models capable of context-aware understanding and a preprocessing pipeline to normalize inputs.

2. Schema Awareness and SQL Mapping

Mapping natural language inputs to SQL queries posed significant difficulties due to the following reasons: Database Schema Complexity, The system had to dynamically adapt to diverse database schemas, including different table structures, column names, and data types. Query Completeness, Ensuring the generated SQL queries covered all aspects of the user query was challenging, especially for complex queries involving joins, nested queries, and aggregate functions. Error Propagation, Minor misinterpretations in the natural language input often led to invalid SQL queries or queries retrieving incomplete results. The implementation of schema-aware mechanisms and careful validation of SQL queries helped mitigate these issues.

3. Reasoning and Explanation

Providing reasoning for both the generated SQL query and the query results added an additional layer of complexity Alignment

of Reasoning with Query, Ensuring that the reasoning accurately reflected the logic behind the generated SQL query was difficult, particularly for advanced queries with multiple conditions or subqueries. Clarity and Coherence,Generating explanations that were both technically accurate and easy for non-technical users to understand was a significant challenge. Handling Empty Results, When the query returned no results, the system needed to generate a meaningful explanation that accounted for the absence of data without confusing the user. Developing a reasoning module with a detailed understanding of SQL operations and user intent required a fine balance between technical accuracy and user comprehension.

4. Integration of Machine Learning Models

The integration of large language models (LLMs) into the system for both query generation and reasoning presented its own challenges Performance and Latency, LLMs are computationally intensive, and ensuring real-time responses without significant delays required optimization of the pipeline. Model Adaptability, Pretrained models were not always perfectly suited for specific database schemas or user intents, necessitating fine-tuning or prompt engineering. Error Handling, In cases where the model generated incorrect or incomplete queries, detecting and correcting these errors programmatically was a major challenge. To address these issues, the system incorporated fallback mechanisms and iterative query refinement techniques.

5. Dataset and Training Challenges

Training the system to handle diverse natural language queries and database schemas required a comprehensive dataset. Data Collection, Curating a dataset that covered a wide range of query types, schemas, and natural language variations was time-consuming.Generalization, Ensuring the model generalized well across unseen schemas and queries was difficult, as overfitting to specific datasets could limit performance. Evaluation Metrics, Defining meaningful metrics to evaluate the accuracy of generated SQL queries and the quality of reasoning explanations posed a significant challenge. These issues were addressed through data augmentation techniques, schema-agnostic training, and careful evaluation metric design.

6. User Experience and Interface Design

Designing a user-friendly interface for the system required overcoming several hurdles.Simplified Interaction, Balancing simplicity with functionality to cater to both technical and non-technical users was challenging. Error Feedback, Providing meaningful feedback to users in case of invalid inputs or errors required robust error-handling mechanisms. Visualization of Results, Presenting complex query results in an intuitive and visually appealing manner posed additional design challenges. Iterative testing and feedback from users helped refine the interface to address these concerns.

7. System Scalability and Maintenance

As the system was designed to handle various datasets and schema structures, scalability and maintainability were critical concerns: Dynamic Schema Adaptation: The system needed to adapt to different database schemas dynamically without requiring manual intervention. Database Size: Handling large databases with millions of records without performance degradation was challenging. Future Model Updates: Integrating updates to the underlying LLMs or database technologies without disrupting the existing workflow required modular design. Addressing these challenges involved designing a flexible architecture with modular components and efficient database query mechanisms.

VIII. Conclusion

In this project, we successfully developed a robust system that bridges the gap between natural language processing and database querying by leveraging advanced AI models and SQL integration. The system empowers users to interact with complex databases through simple, intuitive natural language inputs, automating the generation of SQL queries and providing detailed reasoning for the results. This combination of query generation, execution, and interpretability enhances accessibility for non-technical users while maintaining technical rigor for advanced applications. Throughout the development process, several challenges were addressed, including natural language ambiguity, schema awareness, and integration of machine learning models. The proposed architecture, grounded in modular design and advanced reasoning capabilities, ensures scalability, adaptability, and security, making the system suitable for diverse datasets and use cases. The outcomes of this project demonstrate its potential to revolutionize data interaction by simplifying query processes, reducing dependency on technical expertise, and enabling informed decision-making. This work lays the foundation for further exploration in intelligent query systems, particularly in enhancing reasoning capabilities and expanding domain adaptability.

IX. REFERENCES

[1] Androutsopoulos, I., Ritchie, G. D., & Thanisch, P. (1995). "Natural Language Interfaces to Databases – An Introduction." *Journal of Natural Language Engineering*.

[2] Dong, L., & Lapata, M. (2018). "Coarse-to-Fine Decoding for Neural Semantic Parsing." *Proceedings of the Association for Computational Linguistics (ACL)*.

[3]Finegan-Dollak, C., et al. (2018). "Improving Text-to-SQL Evaluation Methodology." *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

[4]Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why Should I Trust You? Explaining the Predictions of Any Classifier." *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[5] Yu, T., et al. (2018). "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task." *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP)*.

[6] Sebastian Ruder, "An Overview of Multi-Task Learning in Deep Neural Networks," *arXiv preprint arXiv:1706.05098*, 2017.

[7] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, "Distilling the Knowledge in a Neural Network," *Advances in Neural Information Processing Systems*, 2015.

[8] Priyanka Agrawal, Khyati Mahajan, and Vaibhav Kumar, "Eliciting Translation Ability of Large Language Models Using Multilingual Fine-Tuning," *Transactions of the Association for Computational Linguistics*, 2023.

[9] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman, "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding," *Proceedings of ICLR 2019*, 2019.

[10] Goyal, Angela Fan, and Philipp Koehn, "FLORES-101: Evaluating Multilingual Translation Ability," *Proceedings of EMNLP 2021*, 2021.