© OCT 2025 | IRE Journals | Volume 9 Issue 4 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I4-1711081-2290

The Triumph of a Single Programming Language – English

RAUNAK B SINHA
Master's in Data Science and AI, BITS Pilani, India

Abstract- The rapid advancement of Large Language Models (LLMs) and Artificial Intelligence (AI) is catalyzing a fundamental shift in traditional programming paradigms. This paper explores the vision of a future where conventional programming languages such as C++, Java, Python and many more converge into a unified paradigm centered around natural language—primarily English. By introducing an abstraction layer over existing languages, LLMs serve as an intelligent intermediary, enabling users to describe computational logic using natural language instead of formal syntax. This approach eliminates the need for developers to learn and master multiple programming languages and syntaxes, thereby democratizing software development and making programming more accessible. The paper discusses the simple architecture of this superset abstraction, its implications for data analysis, software engineering, and the future of human-computer interaction in natural language.

I. INTRODUCTION

This paper introduces a novel approach to programming that leverages the capabilities of generative AI models to transform natural language instructions directly into executable Python code. We propose a framework where traditional programming languages are abstracted beneath a natural language interface, enabling users to program using plain English, either written or spoken. This is demonstrated through a prototype IPython magic command (%%rb_code) that interprets English-language descriptions of computational logic and executes the AI-generated code in real-time.

By embedding LLMs within the development environment, the proposed system eliminates the need for learning multiple programming syntaxes and offers a highly accessible and intuitive mode of software development. This approach not only democratizes coding for non-programmers but also augments productivity for experienced developers by abstracting boilerplate logic and allowing focus on higher-level problem-solving.

II. RESEARCH AND IDEA

The foundational idea for this research originated from observing the increasing capabilities of Large Language Models (LLMs) such as OpenAI's GPT and Google's Gemini in generating high-quality code from natural language prompts. With the rapid integration of generative AI tools in development environments, a question emerged as below:-
Can programming itself be abstracted to a level where natural language becomes the primary interface, eliminating the need for traditional syntax-based languages?
To validate this, a simple experiment has been performed using Jupyter Notebook for testing , Python Magic Command for using enhanced functionality beyond regular python code , LLM model as Google's Gemini for code generation , Dynamic Memory to make the model stateful & datasets of banking transaction as csv file for data analysis task execution.

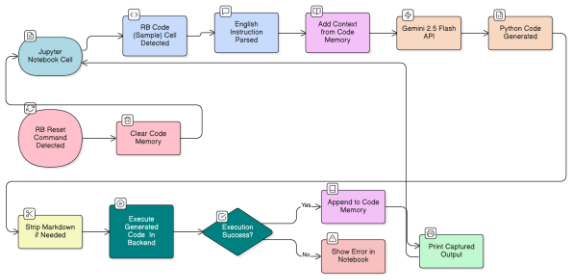
III. DESIGN AND FINDINGS

A. Design

The below design outlines the internal architecture and flow of the custom Jupyter magic command %%rb_code, which allows users to write English instructions that are converted into executable Python code using Google's Gemini 2.5 Flash API. When a notebook cell with the %%rb_code magic is detected, the system parses the English instruction and retrieves previously executed Python code from a global memory store (code_memory). This past context is combined with the new instruction to help Gemini generate more accurate, context-aware Python code.

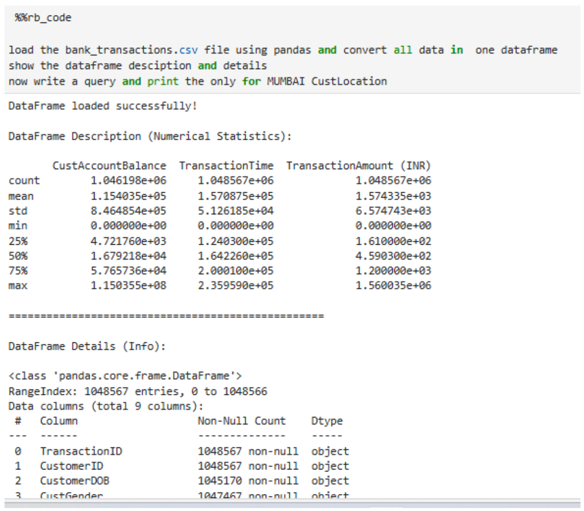
The prompt is then sent to the Gemini 2.5 Flash API, which returns Python code as plain text or markdown. If the output is in markdown format (e.g., inside triple backticks), the markdown is stripped. The resulting code is executed using Python's exec() function within the notebook's global scope. If the code runs successfully, it is appended to code_memory for use in future prompts. If there is an execution error, it is caught and displayed in the notebook output without updating the memory.

Additionally, the design shows the %rb_reset line magic, which clears the stored memory (code_memory), allowing users to reset the assistant's context and begin a new, isolated instruction session



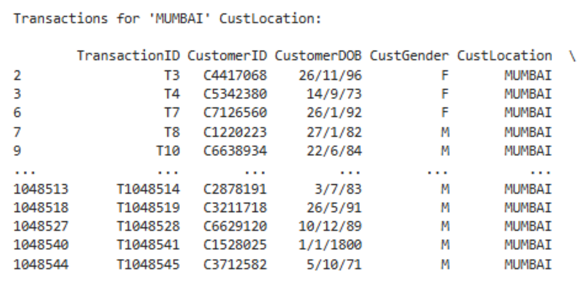
B. Observations and Results

1. Using English language instructions, the system is capable of loading data and performing data preprocessing tasks.



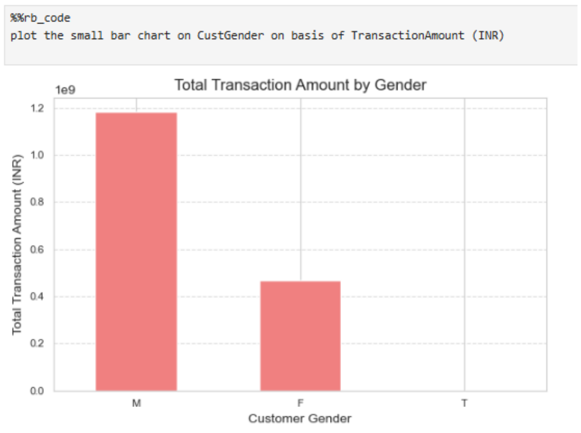
```
%%rb_code

load the bank_transactions.csv file using pandas and convert all data in  one dataframe
show the dataframe desciption and details
now write a query and print the only for MUMBAI CustLocation
```

```
DataFrame loaded successfully!

DataFrame Description (Numerical Statistics):

       CustAccountBalance  TransactionTime  TransactionAmount (INR)
count        1.046198e+06     1.048567e+06             1.048567e+06
mean         1.154035e+05     1.570875e+05             1.574335e+03
std          8.464854e+05     5.126185e+04             6.574743e+03
min          0.000000e+00     0.000000e+00             0.000000e+00
25%          4.721760e+03     1.240300e+05             1.610000e+02
50%          1.679218e+04     1.642260e+05             4.590300e+02
75%          5.765736e+04     2.000100e+05             1.200000e+03
max          1.150355e+08     2.359590e+05             1.560035e+06

==================================================

DataFrame Details (Info):

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048567 entries, 0 to 1048566
Data columns (total 9 columns):
 #   Column                   Non-Null Count    Dtype
---  ------                   --------------    -----
 0   TransactionID            1048567 non-null  object
 1   CustomerID               1048567 non-null  object
 2   CustomerDOB              1045170 non-null  object
 3   CustGender               1047467 non-null  object
```



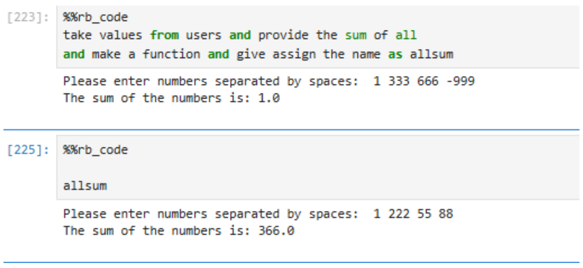
```
Transactions for 'MUMBAI' CustLocation:

        TransactionID CustomerID CustomerDOB CustGender CustLocation  \
2                  T3   C4417068    26/11/96          F       MUMBAI
3                  T4   C5342380     14/9/73          F       MUMBAI
6                  T7   C7126560     26/1/92          F       MUMBAI
7                  T8   C1220223     27/1/82          M       MUMBAI
9                 T10   C6638934     22/6/84          M       MUMBAI
...               ...        ...         ...        ...          ...
1048513      T1048514   C2878191      3/7/83          M       MUMBAI
1048518      T1048519   C3211718     26/5/91          M       MUMBAI
1048527      T1048528   C6629120    10/12/89          M       MUMBAI
1048540      T1048541   C1528025    1/1/1800          M       MUMBAI
1048544      T1048545   C3712582     5/10/71          M       MUMBAI
```

2. Generate plots using English language instructions, as demonstrated in the image below



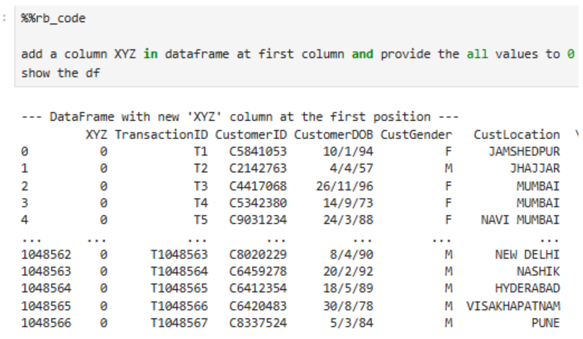
```
%%rb_code
plot the small bar chart on CustGender on basis of TransactionAmount (INR)
```

3. A simple sum function can be created by describing its logic in plain English, without writing any code manually.



```
[223]: %%rb_code
       take values from users and provide the sum of all
       and make a function and give assign the name as allsum

       Please enter numbers separated by spaces:  1 333 666 -999
       The sum of the numbers is: 1.0

[225]: %%rb_code

       allsum

       Please enter numbers separated by spaces:  1 222 55 88
       The sum of the numbers is: 366.0
```

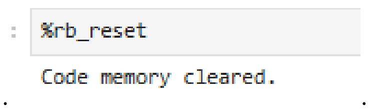
4. A new column can be added to a DataFrame using English language instructions



```
%%rb_code

add a column XYZ in dataframe at first column and provide the all values to 0
show the df
```

```
--- DataFrame with new 'XYZ' column at the first position ---
         XYZ TransactionID CustomerID CustomerDOB CustGender   CustLocation  \
0          0            T1   C5841053     10/1/94          F      JAMSHEDPUR
1          0            T2   C2142763      4/4/57          M         JHAJJAR
2          0            T3   C4417068    26/11/96          F          MUMBAI
3          0            T4   C5342380     14/9/73          F          MUMBAI
4          0            T5   C9031234     24/3/88          F     NAVI MUMBAI
...      ...           ...        ...         ...        ...             ...
1048562    0      T1048563   C8020229      8/4/90          M       NEW DELHI
1048563    0      T1048564   C6459278     20/2/92          M          NASHIK
1048564    0      T1048565   C6412354     18/5/89          M       HYDERABAD
1048565    0      T1048566   C6420483     30/8/78          M   VISAKHAPATNAM
1048566    0      T1048567   C8337524      5/3/84          M            PUNE
```

5. The memory reset process involves clearing all stored data or session information to restore the system to its initial state.



```
%rb_reset

Code memory cleared.
```

IV.CONCLUSION

In this research, we explored a groundbreaking shift in the way humans interact with computers—by using natural language as the medium for programming. With the help of generative AI, specifically Google's Gemini model, we developed a prototype system that allows English instructions to be converted into executable Python code. This experiment, though conducted on a small scale, proves that it is indeed possible to abstract traditional programming languages through natural language. In essence, we have begun the journey of transforming human thought, expressed in everyday language, into functioning software.

This is not just a technical achievement; it represents a broader, human-centered vision of the future of programming. For decades, software development has required individuals to learn complex syntax, logic structures, and multiple languages—each with its own rules and limitations. This creates a steep learning curve and limits access to programming to a small, technically skilled portion of the population. Our work challenges that barrier by proposing a future where anyone, regardless of technical background, can create software simply by describing what they want in English.

Imagine a world where doctors, artists, teachers, or entrepreneurs can build digital tools without writing a single line of code—just by speaking or typing their ideas. As generative AI continues to evolve, we believe this future is not only possible but inevitable.

The journey ahead involves expanding this abstraction to support other languages beyond Python, improving accuracy, handling ambiguity in natural language, and integrating voice-based interaction. But the path is clear: we are moving toward a future where the only programming language needed is the one we already speak.
This research is a meaningful step toward democratizing software development—bringing us closer to a world where programming is no longer just for coders, but for everyone.

REFERENCES

[1] "Data Science and Machine Learning using Python" Author : Dr.Reema Thareja
[2] "Effective Python" Author: Brett Slatkin
[3] "Attention Is All You Need" Authors: Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin
[4] "Speech and Language Processing" Authors: Daniel Jurafsky and James H. Martin