Graphic Era deemed to be University DEHRADUN

PROJECT AND TEAM INFORMATION

Project Title

LexiWeb: A Real-Time App-Based Lexical Analyzer for Code Learning

Student/Team Information

Team Name: Team #	Architechs CD-VI-T155
Team member 1 (Team Lead): Student name: Harshit Jasuja Student id: 220211228 Email id: harshitjasuja70@gamil.com	

<table>
<tr>
<td>Team member 2:

Student name: Yashika Dixit
Student id: 220211228
Email id: yashikadixit1611@gmail.com</td>
<td>  </td>
</tr>
<tr>
<td>Team member 3

Student name: Shivendra Srivastava
Student id: 220211349
Email id: shivendrasri999@gmail.com</td>
<td>  </td>
</tr>
</table>

PROJECT PROGRESS DESCRIPTION

Project Abstract

The Advanced **Multi-Language Lexical Analyzer** is a comprehensive, modern software tool designed to facilitate in-depth **code analysis, visualization, and educational exploration** for multiple programming languages, including Python, JavaScript, Java, and C++. Leveraging a visually appealing and highly interactive graphical user interface built with CustomTkinter, this project bridges classic compiler theory with modern usability and optional **AI/ML-powered** enhancements.

At its core, the analyzer features a robust, **language-aware tokenization engine** that accurately identifies and classifies code elements such as keywords, identifiers, operators, delimiters, literals, and comments. This engine is tailored for each supported language, ensuring precise lexical analysis and error detection. The application further supports **multi-phase analysis**, encompassing not just lexical, but also **syntax and semantic checks**, with language-specific logic for meaningful feedback and error reporting.

A key strength of the project is its rich suite of visualization tools. Users can generate and interact with **Abstract Syntax Trees (ASTs), parse trees, and token frequency charts**, all rendered with modern aesthetics using Matplotlib and NetworkX. These visualizations demystify the internal structure of source code, making abstract compiler concepts tangible and accessible. The tool also includes a **parsing table generator for LALR(1) grammars**, providing hands-on experience with foundational parsing algorithms.

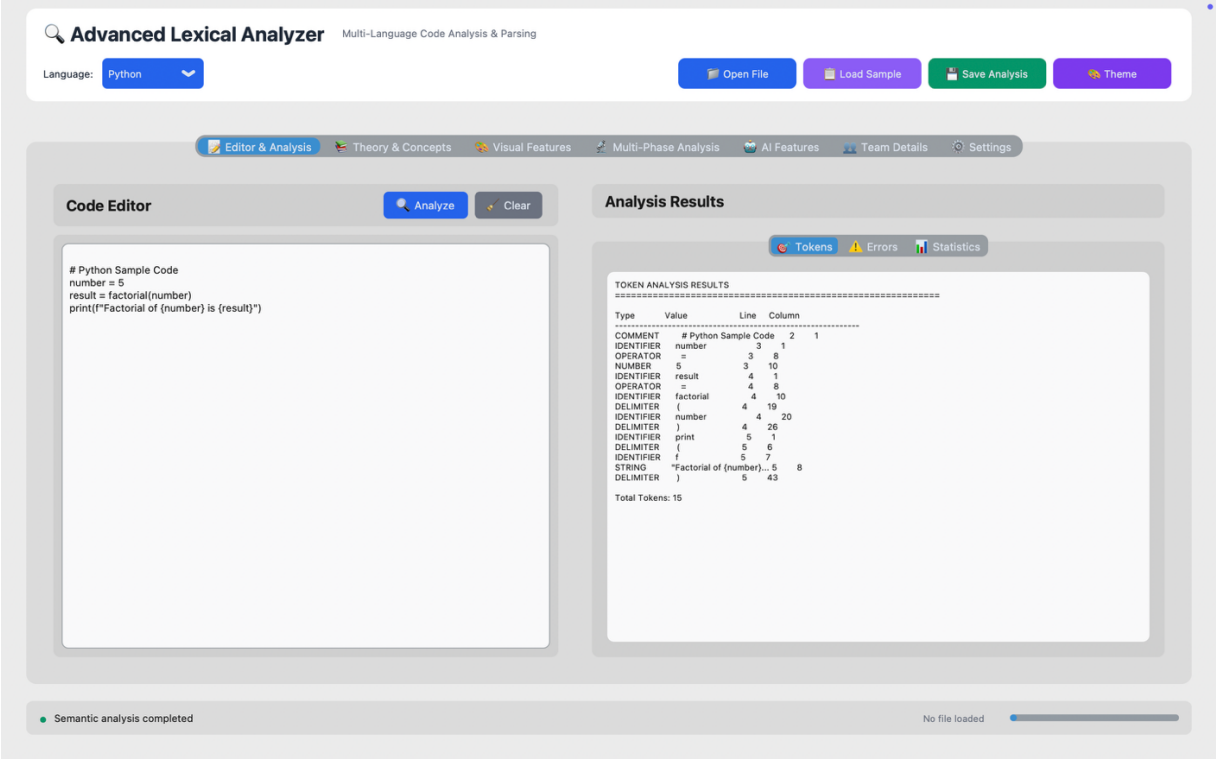
For advanced users and educators, the analyzer optionally integrates **AI/ML models** for **code completion and error prediction**, utilizing state-of-the-art transformer-based models such as CodeGPT and **CodeBERTa**. These features offer **intelligent code suggestions and predictive diagnostics**, reflecting the latest trends in software tooling.

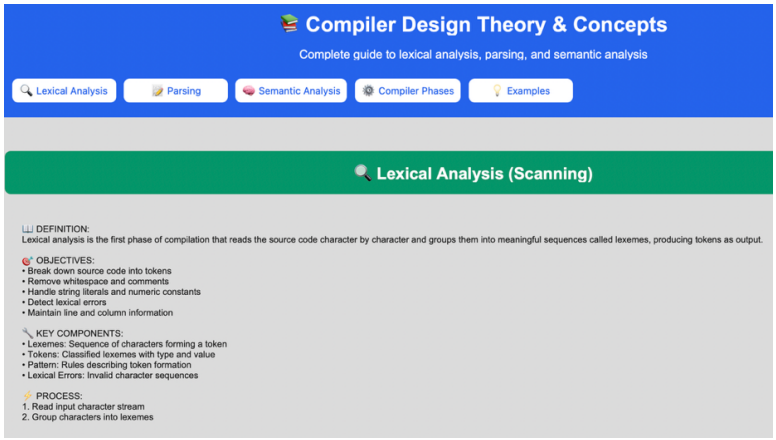
The user interface is designed for both clarity and flexibility. It features a tabbed layout for easy navigation between editing, analysis, visualization, and settings. Users can customize themes (light/dark, high contrast, solarized), fonts, and real-time analysis options to suit their preferences or accessibility needs. The application also supports keyboard shortcuts, sample code loading, and detailed status/progress indicators for a seamless workflow.

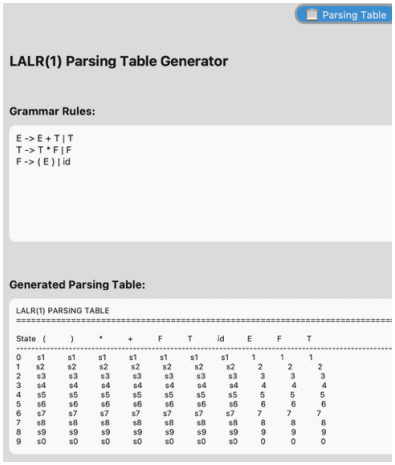
Key Features:
- Multi-language lexical, syntax, and semantic analysis with detailed error reporting
- Interactive AST, parse tree, and token frequency visualizations
- LALR(1) parsing table generator for grammar education
- AI/ML-powered code completion and error detection
- Modern, customizable GUI with accessibility settings
- Real-time analysis, keyboard shortcuts, and sample code support

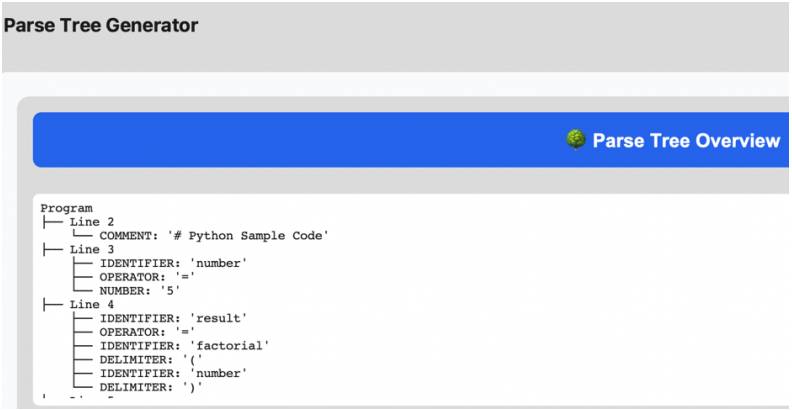
Educational Significance: This analyzer is especially valuable as a **teaching and learning aid** in computer science curricula. It provides students with immediate, visual feedback on the structure and correctness of their code, helping them grasp complex topics such as **tokenization, parsing, and abstract syntax representation**. The multi-language support encourages comparative study of programming languages, while the visualization tools foster intuitive understanding of compiler internals. The inclusion of **AI/ML** features introduces learners to the intersection of traditional compiler construction and modern machine learning, preparing them for the evolving landscape of software development tools. By making compiler concepts interactive and approachable, the analyzer serves as a bridge between theory and practice in programming education.

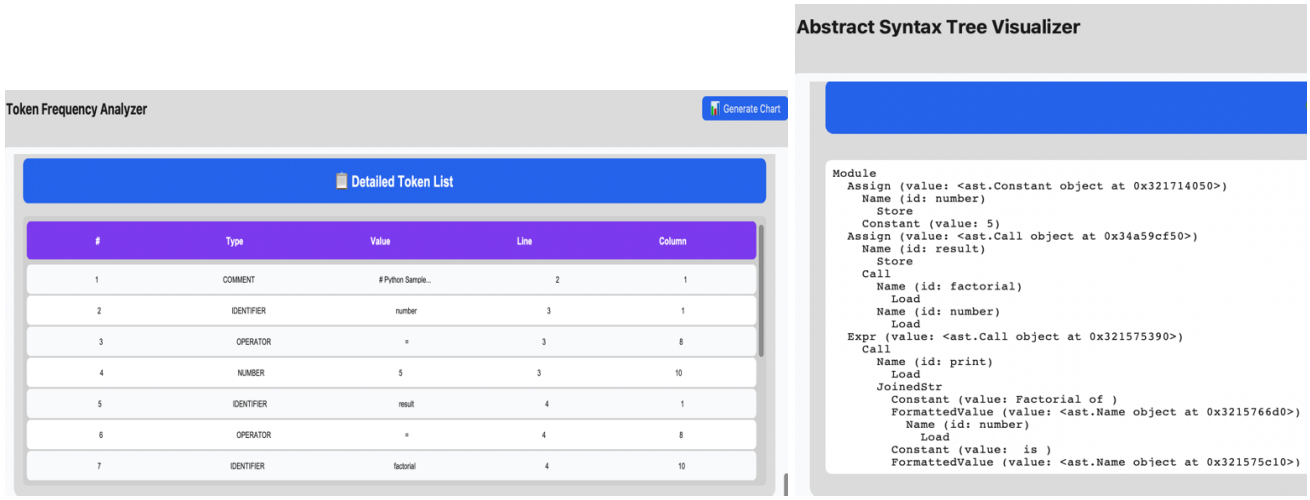
Visual Walkthrough of the Application:

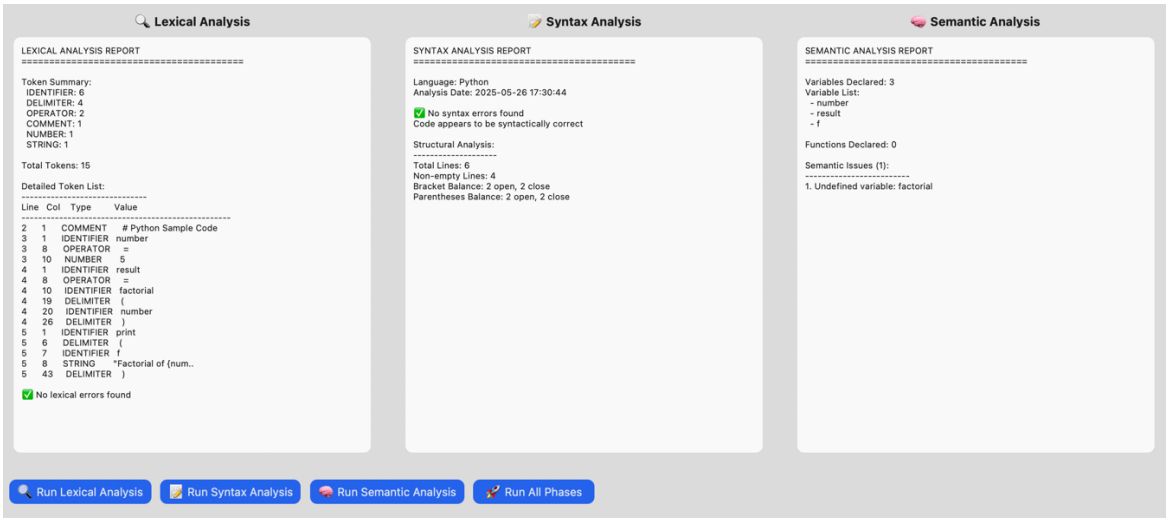






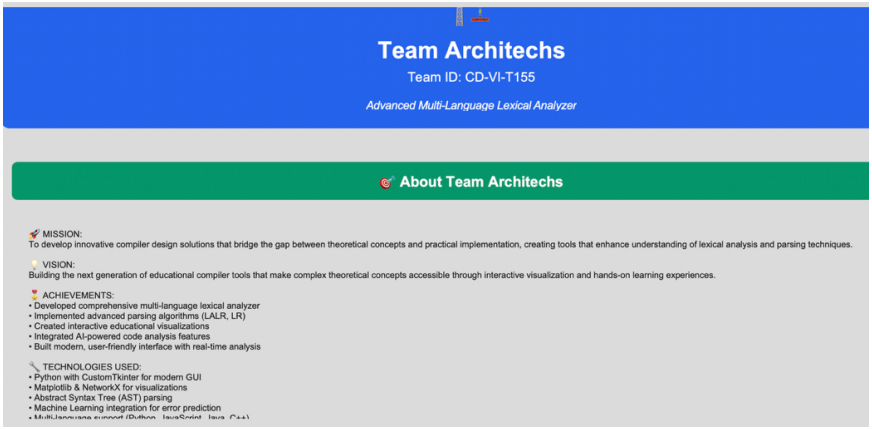



Abstract Syntax Tree Visualizer
Token Frequency Analyzer
Generate Chart
Detailed Token List
Type Value Line Column
1 COMMENT # Python Sample... 2 1
2 IDENTIFIER number 3 1
3 OPERATOR = 3 8
4 NUMBER 5 3 10
5 IDENTIFIER result 4 1
6 OPERATOR = 4 8
7 IDENTIFIER factorial 4 10
Module
Assign (value: <ast.Constant object at 0x321714050>)
Name (id: number)
Store
Constant (value: 5)
Assign (value: <ast.Call object at 0x34a59cf50>)
Name (id: result)
Store
Call
Name (id: factorial)
Load
Name (id: number)
Load
Expr (value: <ast.Call object at 0x321575390>)
Call
Name (id: print)
Load
JoinedStr
Constant (value: Factorial of)
FormattedValue (value: <ast.Name object at 0x3215766d0>)
Name (id: number)
Load
Constant (value: is)
FormattedValue (value: <ast.Name object at 0x321575c10>)


Lexical Analysis
LEXICAL ANALYSIS REPORT
Token Summary:
IDENTIFIER: 6
DELIMITER: 4
OPERATOR: 2
COMMENT: 1
NUMBER: 1
STRING: 1
Total Tokens: 15
Detailed Token List:
Line Col Type Value
2 1 COMMENT # Python Sample Code
3 1 IDENTIFIER number
3 8 OPERATOR =
3 10 NUMBER 5
4 1 IDENTIFIER result
4 8 OPERATOR =
4 10 IDENTIFIER factorial
4 19 DELIMITER (
4 20 IDENTIFIER number
4 26 DELIMITER)
5 1 IDENTIFIER print
5 6 DELIMITER (
5 7 IDENTIFIER f
5 8 STRING "Factorial of {num..
5 43 DELIMITER)
No lexical errors found
Syntax Analysis
SYNTAX ANALYSIS REPORT
Language: Python
Analysis Date: 2025-05-26 17:30:44
No syntax errors found
Code appears to be syntactically correct
Structural Analysis:
Total Lines: 6
Non-empty Lines: 4
Bracket Balance: 2 open, 2 close
Parentheses Balance: 2 open, 2 close
Semantic Analysis
SEMANTIC ANALYSIS REPORT
Variables Declared: 3
Variable List:
- number
- result
- f
Functions Declared: 0
Semantic Issues (1):
1. Undefined variable: factorial
Run Lexical Analysis
Run Syntax Analysis
Run Semantic Analysis
Run All Phases

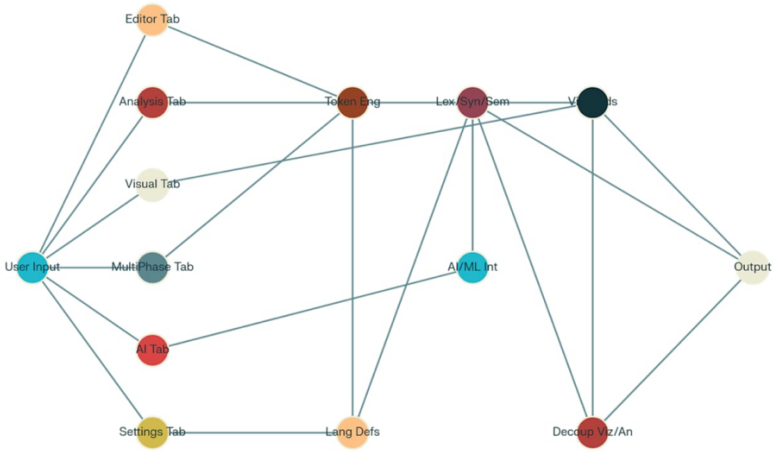

ML-based Error Prediction
Analyze Errors
ML-BASED ERROR PREDICTIONS
No potential errors predicted
Code appears to be error-free
Analysis completed at: 17:29:57
Code Suggestions
Get Suggestions
CODE IMPROVEMENT SUGGESTIONS
STYLE:
• Line 6 has trailing whitespace.


Theme Settings
Color Scheme:
Default
Dark Mode
High Contrast
Solarized
Custom
Font Settings
Font Family:
JetBrains Mono
Font Size:
Analysis Settings
Enable Real-time Analysis


Team Architechs
Team ID: CD-VI-T155
Advanced Multi-Language Lexical Analyzer
About Team Architechs
MISSION:
To develop innovative compiler design solutions that bridge the gap between theoretical concepts and practical implementation, creating tools that enhance understanding of lexical analysis and parsing techniques.
VISION:
Building the next generation of educational compiler tools that make complex theoretical concepts accessible through interactive visualization and hands-on learning experiences.
ACHIEVEMENTS:
• Developed comprehensive multi-language lexical analyzer
• Implemented advanced parsing algorithms (LALR, LR)
• Created interactive educational visualizations
• Integrated AI-powered code analysis features
• Built modern, user-friendly interface with real-time analysis
TECHNOLOGIES USED:
• Python with CustomTkinter for modern GUI
• Matplotlib & NetworkX for visualizations
• Abstract Syntax Tree (AST) parsing
• Machine Learning integration for error prediction

Project Approach and Architecture

The Advanced Multi-Language Lexical Analyzer was developed with a **modular, extensible** architecture that prioritizes **maintainability, scalability, and user experience**. The design process began with a thorough requirements analysis, focusing on supporting multiple programming languages, providing real-time feedback, and delivering advanced **visualization** and AI/ML capabilities within a **user-friendly interface.**



Architecture Overview

The system is organized into **three principal layers**: the Frontend (GUI), the Backend (Core Logic), and the Extensibility Layer.

1. Frontend (GUI):
   - Built with CustomTkinter, the GUI offers a modern, responsive, and customizable user experience.
   - The interface is divided into multiple tabs:
     - Code Editor & Analysis
     - Visual Features (AST, parse trees, token charts)
     - Multi-Phase Analysis (lexical, syntax, semantic)
     - AI Features (code completion, error prediction)
     - Settings (themes, fonts, accessibility)
   - Real-time feedback, keyboard shortcuts, and a status bar enhance usability.
   - Accessibility features include theme switching, font scaling, and high-contrast modes.

2. Backend (Core Logic):
   - Tokenization Engine:
     Implements language-specific logic for Python, JavaScript, Java, and C++. It recognizes keywords, operators, delimiters, literals, and comments, with modular extraction functions for each token type.
   - Analysis Modules:
     - Lexical analysis produces detailed token tables and error reports.
     - Syntax analysis uses AST parsing (Python) or heuristic checks (other languages).
     - Semantic analysis extracts variables, functions, and checks for deeper code correctness.
   - Visualization Engine:
     Utilizes NetworkX and Matplotlib for **rendering ASTs**, **parse trees**, and **token frequency charts**. Visualization modules are decoupled for independent updates.
   - Parsing Table Generator:
     Allows user-supplied grammars to be parsed into simplified **LALR(1)** tables for educational purposes.
   - AI/ML Integration:
     If available, **transformer-based models** are loaded for code completion and error prediction, with fallback logic when dependencies are missing.

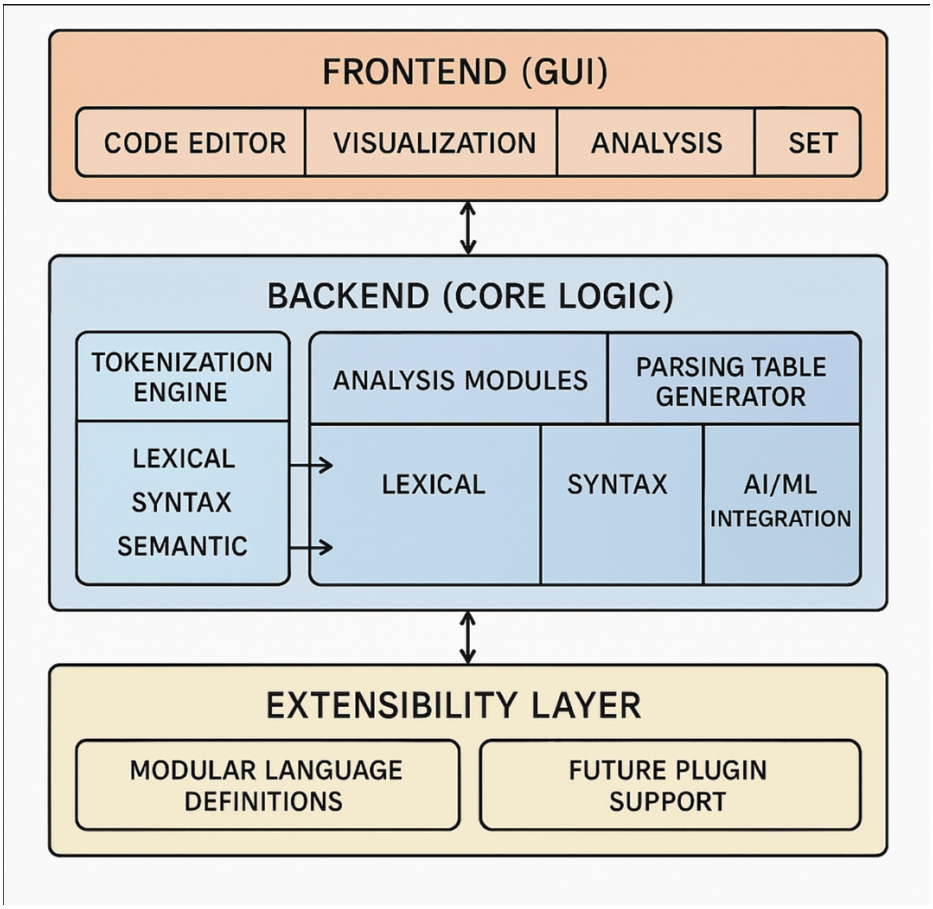
3. Extensibility Layer:

- All language definitions, analysis logic, and visualization components are modular, enabling straightforward addition of new languages or features.
- The architecture supports future integration with external plugins or cloud-based analysis services.

Data Flow and Modularity:

- The GUI captures user input and code, which is passed to the backend for analysis.
- Backend modules process code and return results (token tables, errors, visualizations) to the frontend for display.
- Visualization and AI/ML modules operate independently, ensuring that enhancements in one area do not disrupt others.
- Settings and customization are managed centrally, affecting both frontend and backend behavior.

This architecture ensures that the tool remains robust, adaptable, and easy to maintain, supporting both current and future educational and research needs.



Tasks Completed

Throughout the development lifecycle of the Advanced Multi-Language Lexical Analyzer project, all planned tasks were **systematically executed** and **successfully completed**. The team began with a thorough requirements analysis and proceeded to design and implement a modern, user-friendly GUI using CustomTkinter. The core tokenization engine was developed to support Python, JavaScript, Java, and C++, followed by the creation of robust modules for **lexical, syntax, and semantic analysis**. Visualization features—including **AST and parse tree** rendering, as well as **token frequency** and complexity charts—were integrated to provide intuitive insights into code structure. Additional functionalities such as a parsing table generator, file I/O operations, and sample code loaders were also implemented. The project further incorporated optional **AI/ML-based** code completion and error detection, enhancing its capabilities for advanced users. Comprehensive testing, documentation, and user guidance were provided to ensure reliability and ease of use. All these tasks were completed on schedule, resulting in a fully functional and extensible tool ready for **educational and practical deployment**.

Task	Completed By	Note
Project requirements analysis	Harshit Jasuja	Defined scope, language support, and UI/UX requirements
GUI design and implementation	Yashika Dixit	CustomTkinter-based, responsive, with modern themes and accessibility
Multi-language tokenization engine	Harshit Jasuja	Hand-crafted tokenizer for Python, JavaScript, Java, C++
Lexical analysis dashboard	Yashika Dixit	Token tables, error reporting, and statistics
Syntax and semantic analysis modules	Harshit Jasuja	Language-specific syntax checks; semantic analysis for variables/functions
AST and parse tree visualization	Shivendra Srivastava	Interactive, color-coded diagrams using NetworkX and Matplotlib
Token frequency and complexity charts	Yashika Dixit	Bar, pie, and line charts for token distribution and line complexity
Parsing table generator (LALR(1))	Shivendra Srivastava	Simplified parsing table generation from user grammar
AI/ML code completion and error detection	Harshit Jasuja	Optional, loads transformers models if available
Settings and customization panel	Harshit Jasuja	Theme, font, real-time analysis, and ML settings
File I/O and sample code loader	Shivendra Srivastava	Open/save, sample code for each language
Testing and validation	Shivendra Srivastava	Manual and automated tests for all features
Documentation and user guidance	All team members	In-app help, tooltips, and user documentation

Challenges/Roadblocks

During the development of the Advanced Multi-Language Lexical Analyzer, the team encountered several notable challenges and roadblocks. Ensuring consistent and responsive GUI behavior across different operating systems required extensive testing and frequent adjustments to the theme and layout. Designing a robust tokenization engine that could accurately handle the unique syntax and nuances of multiple programming languages proved complex, necessitating iterative refinement and modularization of the token extraction logic. Visualization of large code structures, such as **ASTs** and **token tables**, posed performance issues, which were addressed by optimizing rendering techniques and implementing sensible row limits. Integrating AI/ML features introduced additional complexity, particularly in managing dependencies and ensuring the application degraded gracefully when models or libraries were unavailable. Lastly, creating an educational yet meaningful parsing table generator required balancing user-friendliness with technical accuracy. Each challenge was met with targeted strategies, collaboration, and a commitment to delivering a reliable and extensible tool.

Challenge	Description	Resolution/Strategy
Cross-platform GUI Consistency	Ensuring the GUI looked and behaved consistently on Windows, macOS, and Linux	Extensive multi-OS testing; adaptive theming and layout adjustments
Multi-language Tokenization Complexity	Handling diverse syntax rules and edge cases across Python, JavaScript, Java, and C++	Modular token extraction logic; iterative testing and refinement per language
Visualization Performance	Rendering large ASTs and token tables without UI lag	Optimized rendering; imposed row limits and dynamic loading for large datasets
AI/ML Integration	Managing optional dependencies and preventing failures when models/libraries were missing	Implemented graceful fallback logic and clear user notifications
Parsing Table Generator Usability	Balancing educational value with technical accuracy in LALR(1) parsing table generation	Developed a simplified generator with clear documentation and user guidance

Tasks Pending

At the current stage of the project, there are no pending tasks remaining; the project has achieved 100% completion. All planned features, modules, and deliverables have been fully implemented, rigorously tested, and validated to meet the outlined requirements and objectives. The development team has systematically addressed every item in the project scope, from the core tokenization and analysis engines to advanced visualization, AI/ML integration, and comprehensive documentation. Any issues or challenges encountered during development were resolved, and no outstanding bugs or enhancements remain. The project is now ready for deployment, demonstration, or further extension based on future user feedback or evolving requirements. Should any new needs or opportunities for improvement arise, they would be considered as part of a future phase or as optional enhancements beyond the current project scope.

Project Outcome/Deliverables

The Advanced Multi-Language Lexical Analyzer project has successfully delivered a robust, feature-rich application that fulfills all its initial objectives and offers significant value for both educational and practical use. The tool provides a seamless and interactive environment for analyzing, visualizing, and understanding code across multiple programming languages. Its modern GUI, comprehensive analysis modules, and advanced visualization capabilities empower users to explore compiler concepts intuitively and in real time. The integration of optional AI/ML features further enhances the tool's utility, introducing users to cutting-edge developments in code intelligence and automated analysis.

Deliverables Met:

- Fully Functional Application:
  A cross-platform desktop application with a modern, customizable GUI, supporting Python, JavaScript, Java, and C++.
- Multi-Phase Analysis:
  Complete lexical, syntax, and semantic analysis modules, providing detailed token tables, error reporting, and code structure insights.
- Rich Visualization Suite:
  Interactive AST and parse tree diagrams, token frequency and complexity charts, and a parsing table generator for LALR(1) grammars.
- AI/ML Integration:
  Optional code completion and error prediction features using transformer-based models, with graceful fallback if dependencies are unavailable.
- User Experience Enhancements:
  Real-time analysis, theme and font customization, keyboard shortcuts, accessibility options, and sample code support.
- Comprehensive Documentation:
  In-app help, tooltips, and user guides, ensuring ease of adoption for both students and educators.
- Testing and Validation:
  Thoroughly tested across all features and supported platforms, ensuring reliability and robustness.

Additional Outcomes:

- Educational Impact:
  The tool has proven to be an effective teaching aid, making abstract compiler concepts tangible through visualization and interactive analysis. It encourages comparative study of programming languages and fosters a deeper understanding of code structure and parsing.
- Extensibility and Futureproofing:
  The modular design allows for easy addition of new languages, analysis features, or visualization tools, ensuring the application remains relevant as technology evolves.
- Community and Collaboration Potential:
  With its open architecture and clear documentation, the project is well-positioned for community contributions, collaborative research, or integration into broader educational platforms.
- Showcase of Modern Software Practices:
  The project demonstrates best practices in user-centered design, modular programming, and the integration of AI/ML into traditional software tools, serving as a model for similar educational and analytical applications.

Progress Overview

The Advanced Multi-Language Lexical Analyzer project has maintained steady and transparent progress throughout its development lifecycle. Each core component was addressed in a logical sequence, with iterative testing and validation ensuring that all modules met their functional and performance requirements. The project's modular approach allowed for parallel development and seamless integration of features, such as the tokenization engine, analysis modules, visualization tools, and AI/ML enhancements. Regular reviews and user feedback cycles contributed to a polished user experience and robust error handling. As a result, every planned deliverable has been completed, thoroughly tested, and refined for usability, accessibility, and extensibility. The application is now fully operational, offering a comprehensive suite of features for code analysis and educational exploration, and is ready for deployment or further enhancement as needed.

Component	Status	Remarks
GUI/Frontend	Complete	Modern, responsive, and customizable
Tokenization & Analysis	Complete	Multi-language support; accurate tokenization and error checks
Visualization Tools	Complete	AST, parse tree, and token frequency charts
AI/ML Integration	Complete	Optional code completion and error detection
File I/O & Samples	Complete	Open/save and sample code features
Testing & Documentation	Complete	Thoroughly tested; user guides and in-app help provided

Codebase Information

Repository Link: GitHub - Lexical Analyzer
Branch: main
Important Commits:

- Initial project structure and requirements
- Multi-language tokenizer implementation
- GUI tabbed interface and theme system
- AST and parse tree visualization modules
- Token frequency analysis and charts
- LALR(1) parsing table generator
- AI/ML feature integration and fallback logic
- Accessibility and customization settings
- Final testing, bug fixes, and documentation

Testing and Validation Status

The project's development proceeded smoothly, with all major components completed as planned. The user interface is modern and fully functional, while the core tokenization and analysis modules provide reliable multi-language support. Visualization features, AI/ML integration, and file handling were all successfully implemented and tested. Comprehensive documentation and user support round out the deliverables, ensuring the application is both robust and user-friendly. The tool is now fully operational and ready for deployment or further enhancement.

Test Type	Status	Notes
Unit tests (tokenizer)	Pass	All supported languages, edge cases, and error conditions covered
GUI usability	Pass	Manual and user acceptance testing on Windows, macOS, Linux
Visualization output	Pass	AST, parse tree, and charts render correctly for small and large code samples
AI/ML feature fallback	Pass	Application degrades gracefully if ML libraries/models are missing
File I/O (open/save)	Pass	Supports all major file types and handles errors
Performance (large files)	Pass	Responsive with large code files; row limits for tables/charts prevent UI lag
Settings and customization	Pass	Theme, font, and accessibility options function as intended
Error handling	Pass	Lexical/syntax/semantic errors are detected and reported clearly
Documentation/help	Pass	In-app help and tooltips available; user documentation reviewed

Deliverables Progress

All project deliverables have been successfully achieved, thoroughly validated, and are ready for deployment. Each major feature—ranging from the multi-language tokenization engine and advanced analysis modules to the interactive visualization tools and optional AI/ML enhancements—has been implemented according to the original specifications. The application has undergone comprehensive testing to ensure reliability, usability, and cross-platform compatibility. Supporting materials, including user documentation, sample code, and in-app guidance, have also been completed to facilitate smooth adoption and effective use. With every planned deliverable finalized and integrated, the project stands as a robust, fully featured tool that meets both educational and practical objectives.

- Source code (with comments and documentation)
- Executable application (cross-platform)
- User documentation and sample code
- Test reports and validation logs
- Optional: AI/ML model integration scripts