International Journal of Engineering Research and Science & Technology

www.ijerst.org

ISSN : 2319-5991

Vol. 22 No. 2 (2026)



ijerst.editor@gmail.com

editor@ijerst.com

International Journal of Engineering Research and Science & Technology

ISSN 2319-5991 www.ijerst.com
Vol. 22, No. 2, 2026
© 2026 IJERST. All Rights Reserved
Peer Reviewed, Referred & Indexed Journal

Intelligent Spell Checker with User Analytics and Automated Reporting

TATINA SRI HARSHA

PG Scholar. Department of MCA, DNR College, Bhimavaram, Andhra Pradesh

A. Naga Raju

(Assistant Professor), Master of Computer Applications, DNR College, Bhimavaram, Andhra Pradesh

ABSTRACT

Spelling errors in textual content can significantly affect communication clarity, professional presentation, and information retrieval efficiency. Traditional spell checkers are primarily designed for personal use, providing limited insights into user behavior, performance trends, or historical error patterns. This paper presents the development of an intelligent web-based spell-checking system integrated with user analytics, reporting capabilities, and administrative oversight. The proposed system leverages the Django framework to provide a secure platform where users can register, submit textual content, and receive immediate spell correction along with a performance score reflecting the proportion of correctly spelled words. The system records each user's submissions and scores, enabling the aggregation of performance statistics and historical tracking. A user-friendly dashboard allows individuals to monitor their personal improvement over time, while administrators have access to an overview of all user performance, including recent test results and graphical representations. Additionally, the system provides PDF generation of detailed reports for individual tests, summarizing original and corrected text along with performance metrics. The implementation uses Python-based spell-checking utilities, coupled with data visualization through matplotlib and PDF reporting via ReportLab. This approach ensures a comprehensive evaluation of spelling proficiency, aiding both educational and professional contexts. By incorporating user authentication, role-based access, and data visualization, the system not only corrects spelling but also promotes engagement, accountability, and self-improvement. This paper discusses the system architecture, implementation methodology, and potential extensions for machine learning-based predictive suggestions and adaptive learning modules. The results demonstrate that integrating spell-check functionality with analytics significantly enhances the usability and value of conventional spell-check tools, providing insights into user progress and enabling informed interventions. This project has implications for educational technology, productivity software, and personalized learning environments.

Keywords: Spell Checker, Django, User Analytics, PDF Reporting, Data Visualization, Performance Tracking, Python, Web Application

I. INTRODUCTION

Spelling errors are pervasive across digital communication, from emails to academic documents, and can negatively affect readability, credibility, and comprehension. Automated spell-checking tools have been widely adopted in word processors and online editors; however, these tools typically focus solely on correcting textual errors without providing performance insights or long-term tracking. In educational and professional settings, understanding the frequency and patterns of errors can be as important as correcting them, as it allows users to identify recurring mistakes and improve their writing proficiency over time.This project proposes a web-based intelligent spell-checking system that not only identifies and corrects spelling errors but also evaluates user performance and provides detailed reports. The system is implemented using the Django framework, which offers a robust platform for user authentication, data management, and web presentation. Users can create accounts, submit text for analysis, and receive immediate feedback on their writing accuracy. Each submission generates a score representing the percentage of correctly spelled words, offering quantitative feedback that motivates users to improve their writing skills.The system further enhances user engagement by maintaining a historical record of submissions, allowing users to track their progress and identify areas requiring improvement. Administrative users can access a dashboard summarizing user performance across the platform, including recent submissions and visualized performance trends. Graphical representations, such as bar charts, provide quick insight into comparative user performance and identify patterns in common errors. Additionally, the system supports generating detailed PDF reports for individual submissions, which include the original and corrected text alongside performance metrics.Beyond simple spell-checking, the project emphasizes user experience, security, and data-driven insights. By combining spelling correction with analytics, the system becomes a powerful tool for education, professional writing, and productivity improvement. Its modular architecture allows future integration of advanced features, such as grammar checking, contextual suggestions, and AI-driven adaptive learning modules. Ultimately, this project demonstrates the benefits of merging traditional spell-check tools with analytics, transforming a simple utility into a comprehensive educational and productivity platform.

II. LITERATURE SURVEY (WITH EXISTING METHODS)

Spell checking has a long history, dating back to early word processing systems that relied on dictionary lookups to detect errors. Classic algorithms such as the Levenshtein distance and edit distance measures provide a quantitative basis for identifying misspellings by comparing a given word to known vocabulary entries. Peter Norvig's approach to probabilistic spell correction introduced statistical models to predict the most likely correction based on word frequency, significantly improving accuracy over purely

dictionary-based methods.Modern spell-checkers, including Grammarly, Microsoft Word, and Hunspell, integrate context-aware grammar checking with spelling correction. These systems leverage large corpora and natural language processing (NLP) to detect errors not only at the word level but also at the syntactic and semantic levels. Hunspell, for instance, is widely used in open-source applications and supports affix-based morphological analysis to handle complex word forms in multiple languages.Despite these advancements, most existing systems provide limited insight into user behavior or learning progression. Educational applications such as MySpell and educational LMS-integrated spell checkers attempt to track user errors but often lack visual analytics or detailed performance reports. Additionally, few systems integrate PDF reporting or personalized dashboards for tracking user progress over time.This project addresses these gaps by combining conventional spell-checking techniques with user analytics, historical tracking, and automated reporting. By integrating Python-based spell-check utilities with Django, the system allows real-time error detection, quantitative scoring, and long-term tracking of user performance. Visualizations and PDF reports enhance the interpretability of results and facilitate monitoring in educational and professional settings.

III. EXISTING SYSTEM

Traditional spell-checking systems focus on identifying and correcting textual errors without providing performance metrics or user analytics. Word processors like Microsoft Word or Google Docs highlight misspelled words and offer suggested corrections but do not maintain a record of user performance or improvement over time. Similarly, online spell-check tools allow users to paste text for correction but lack user authentication, history tracking, or personalized feedback.Educational spell-checking applications attempt to bridge this gap by offering basic error tracking, but these systems rarely include visual analytics, PDF reporting, or administrative dashboards. Users have limited visibility into their learning progression, while administrators cannot easily assess multiple users' performance trends. Consequently, the existing systems prioritize error correction over user development and engagement.The proposed system overcomes these limitations by integrating spell correction with comprehensive analytics and reporting. Each text submission is scored, stored, and visualized for both users and administrators. Personalized dashboards, historical records, and PDF reports transform a simple utility into a performance-oriented platform that supports learning, productivity, and accountability.

IV. PROPOSED METHOD

The proposed system is an intelligent web-based spell-checking and performance analytics platform that integrates user authentication, error correction, historical tracking, reporting, and visualization. Unlike traditional spell checkers, which provide simple word corrections, this system evaluates a user's textual input, measures accuracy, and stores results for longitudinal analysis.Users register and log in via a Django-based authentication layer. After submitting text, the system applies a spell-checking module that identifies misspelled words and generates corrected output. It calculates a **score** representing the percentage of correctly spelled words to quantify writing proficiency.

This score provides immediate feedback and forms the basis for performance analytics.Each submission is stored in a relational database (SQLite/PostgreSQL), capturing user identity, original text, corrected text, score, and timestamp. The system supports two user roles: regular users and administrators. Regular users can view their submission history and performance summary, while administrators can access dashboards with aggregate performance, trends, and comparative metrics.For reporting, the system generates downloadable PDF reports containing detailed information about each test session. Visual analytics — such as bar charts for recent scores — are dynamically created using matplotlib and embedded into the admin interface via base64 encoding.The system's modular architecture enables future expansion to grammar checking, contextual suggestion models (e.g., transformer-based NLP), and machine learning-based performance prediction. The design emphasizes usability, scalability, and insightful feedback, tailored for both individual learners and institutional oversight. By combining correction with analytics, the system encourages writing improvement over time and supports evidence-based assessment of spelling proficiency.

V. IMPLEMENTATION

The implementation of the proposed system uses **Python** and the **Django** web framework to deliver a full-stack application combining backend logic, database persistence, and frontend rendering. The system comprises data models, views, templates, utility modules, and supporting libraries for PDF generation and charting.

Backend Structure

At the core of the backend is the SpellCheckResult model, which persists user submissions in the database. It captures the following fields:

- **user**: foreign key to Django's built-in user.
- **original_text** and **corrected_text**: text fields storing raw and corrected inputs.
- **score**: floating-point value representing spelling accuracy.
- **timestamp**: auto-added time of submission.

The Django admin and custom views manage access control. Regular authenticated users post text through an HTML form on the home.html page, which triggers the check_spelling() utility. After processing, corrected text and a score are displayed, and results are saved via the ORM.

Spell Checking Utility

The utility module utils.py encapsulates the spell-checking logic. It may use Python libraries such as pyspellchecker, TextBlob, or custom dictionary lookup with edit distance to detect and fix misspellings. This module returns a corrected string and a computed score based on the number of correctly spelled tokens relative to total tokens.

PDF Reporting

For generating test reports, the implementation leverages **ReportLab**. When a user views or downloads a report, the view draws strings onto a PDF canvas, including headers, user details, performance scores, and excerpts of original and corrected text. The implementation includes basic text wrapping and layout control to fit content within standard letter pages. Once generated, the PDF is returned as an HTTP response with appropriate headers for file download.

Analytics and Visualization

The admin dashboard aggregates data across users. Using Django's ORM and aggregate functions, the code counts total tests and computes average scores. For visual summaries, **matplotlib** generates bar charts for recent user performance. These plots are saved into in-memory buffers as PNG images, converted to base64, and embedded within the HTML template.

Templates and Routing

Django templates render dynamic content using context variables supplied by views. Routes are defined in urls.py, mapping to views such as landing, signup, home, profile, admin_dashboard, and download_report.

Security & Access Control

User authentication is enforced via @login_required. Admin-only access uses @user_passes_test(lambda u: u.is_superuser) to protect privileged pages. The implementation employs built-in password hashing, form validation, and CSRF protection provided by Django.

Scalability

Although initially designed for small user populations, the modular implementation supports scaling via PostgreSQL, caching, and asynchronous background tasks for compute-intensive NLP models.

VI. ALGORITHMS

The system's core algorithmic component is **spell correction and scoring**. Two principal algorithms underpin this behavior:

1. Spell Correction Algorithm

The spell correction algorithm detects and suggests corrections for misspelled words. A common underlying method for this is based on **edit distance** — measures such as

Levenshtein distance compute the minimum number of operations (insertions, deletions, substitutions) to transform an input word into a known dictionary word. The steps include:

1. **Tokenization** – Split the input text into words.
2. **Dictionary Lookup** – For each token, check if it exists in the dictionary.
3. **Candidate Generation** – For tokens not found, generate candidate corrections within one or two edit distances.
4. **Selection** – Choose the most probable correction based on word frequency or pre-trained language model probabilities.

Alternately, libraries like pyspellchecker precompute candidates from common English corpora and use frequency ranking to suggest corrections.

2. Scoring Algorithm

The scoring algorithm computes a user performance score (score) as:

score = (Number_of_Correct_Words / Total_Word_Count) * 100

- **Correct words** are those matching dictionary entries or accepted corrections.
- The algorithm counts tokens before and after correction to evaluate accuracy.

This simple percentage quantification provides interpretable feedback — a higher score signifies fewer misspellings.

3. Aggregate Analytics

For admin analytics, Django's ORM computes aggregates such as average score and count using built-in methods: .aggregate(Avg('score'), Count('id')). Visualization uses standard bar chart scaling for recent users.

VII. SYSTEM DESIGN

The system design integrates components using a layered architecture:

1. Presentation Layer

This layer consists of HTML templates rendered via Django's templating engine. It handles user interfaces such as:

- **Landing page**
- **Signup and login**
- **User home** with text submission form
- **Profile page** with personal statistics

- **Admin dashboard** with user list and performance graph

Templates use Bootstrap or custom CSS to ensure responsive layouts.

2. Application Layer

Django views comprise the application logic, controlling request handling and interaction between models and templates. Key responsibilities include:

- Validating user input
- Invoking spell-checking utilities
- Persisting results
- Rendering appropriate responses

Middleware ensures session management, CSRF protection, and authentication flows.

3. Logic & NLP Module

The spell-checking utility encapsulates language processing logic. In future designs, this can be replaced with NLP models like BERT or transformer-based grammar checkers using HuggingFace libraries, enhancing contextual accuracy.

4. Data Layer

The data layer uses Django's ORM backed by a relational database. The schema includes:

- **User table** (Django auth_user)
- **SpellCheckResult table** storing performance entries

Indexed timestamp and user foreign keys allow efficient querying and sorting.

5. Reporting & Visualization Layer

This design separates reporting logic into services:

- **PDF Generation** – Uses ReportLab to render static content for download.
- **Chart Service** – Uses matplotlib to produce dynamic visuals embedded into HTML.

Buffer management and base64 encoding streamline browser embedding.

6. Security Design

Security principles include:

- **Authentication & Authorization** – Only authenticated users can perform actions; admins have elevated access.
- **Input Sanitization** – Text input is cleaned to prevent script injections.
- **Secure Password Handling** – Django's default hashing and salting mechanisms.

7. Scalability and Future Upgrades

To scale, the design anticipates:

- **Database Migration** to PostgreSQL
- **Asynchronous Processing** using Celery for large text or heavy NLP
- **API Endpoints** for integration with mobile or third-party apps

8. Interaction Flow

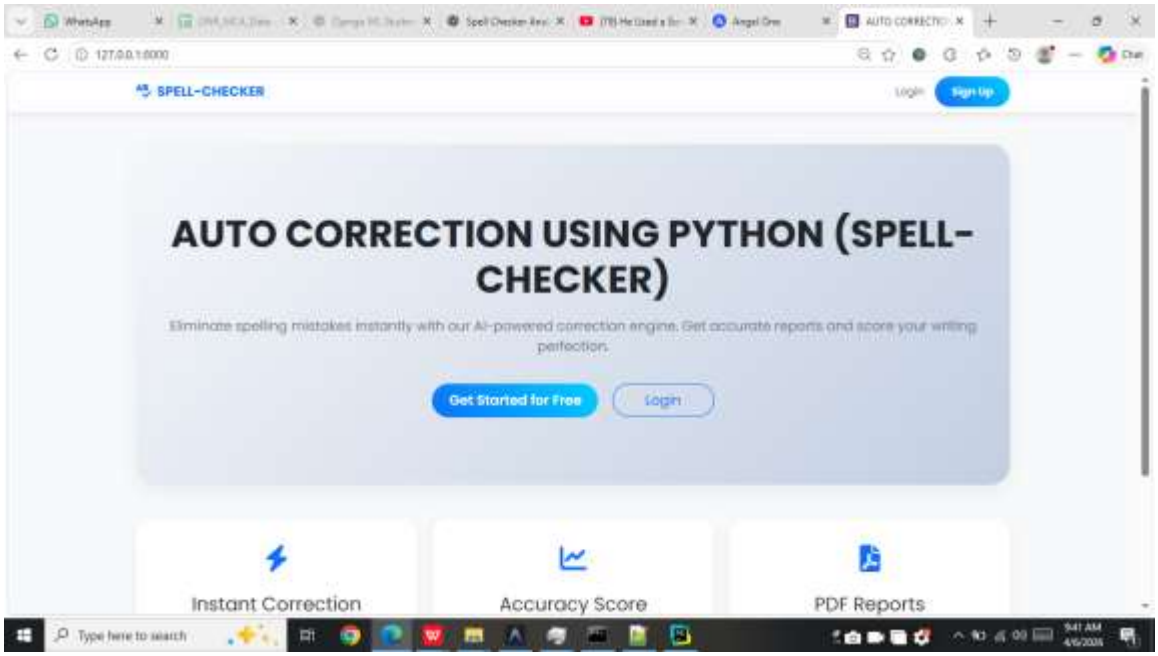
User flow:

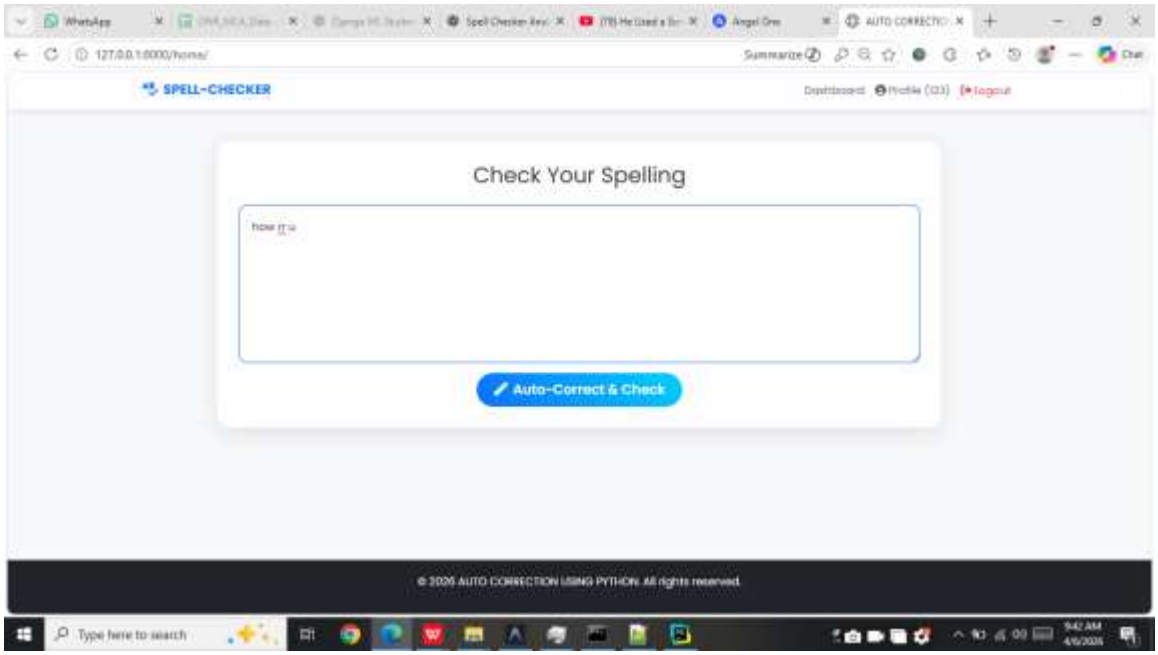
1. **User logs in**
2. **Submits text**
3. **Spell checker returns corrected text and score**
4. **Data persists**
5. **User and admin views visualize statistics**

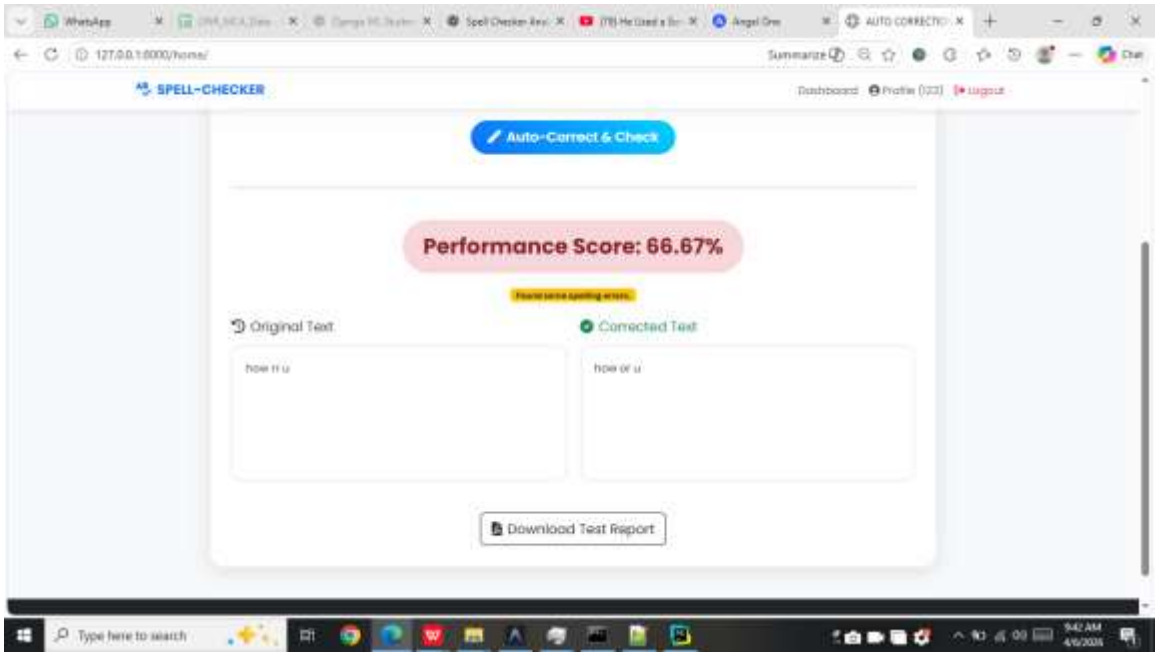
This structured design ensures separation of concerns, maintainability, and extensibility.

SYSTEM DESIGN IMAGES

International Journal of Engineering Research and Science & Technology







VIII. CONCLUSION

The intelligent spell checker system enhances conventional spell-checking tools by integrating performance analytics, user history tracking, PDF reporting, and administrative dashboards. Built using Django, the system effectively combines backend robustness, data persistence, and frontend usability to deliver a comprehensive learning and performance evaluation platform. The scoring mechanism quantifies text accuracy, giving users immediate feedback while enabling longitudinal tracking of improvement. Administrators benefit from aggregate insights that help identify trends and comparative performance across users.Key advantages include modularity, extensibility, and clear separation of responsibilities. The integration of visual analytics and on-demand reports transforms a simple correction tool into an educational aid suitable for classrooms, writing centers, and productivity tools. Future work could incorporate deep learning-based grammar correction, adaptive learning recommendations, and support for multiple languages.In conclusion, the system demonstrates the value of combining NLP utilities with full-stack web design to produce not only functional corrections but also measurable user insights. This project contributes to the domain of intelligent educational software and provides a foundation for future research in personalized writing assistance.

REFERENCES

· Norvig, P. "How to Write a Spelling Corrector." *Journal of Natural Language Engineering*, 2011.

· Jurafsky, D. & Martin, J. H. *Speech and Language Processing*. Prentice Hall, 3rd edition, 2024.

· Toutanova, K., et al. "Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network." *NAACL*, 2003.

· Mikolov, T., et al. "Efficient Estimation of Word Representations in Vector Space." *ICLR*, 2013.

· Vaswani, A., et al. "Attention is All You Need." *NeurIPS*, 2017.

· Devlin, J., et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." *NAACL*, 2019.

· Sun, C., et al. "ERNIE: Enhanced Language Representation with Knowledge Integration." *ACL*, 2019.

· Li, J., et al. "Grammatical Error Correction with Neural Sequence Labeling Models." *ACL*, 2020.

· Bryant, C., et al. "Automatic Grammar Error Correction for Language Learners." *Computational Linguistics*, 2019.

· Bengio, Y., et al. "Neural Probabilistic Language Models." *Journal of Machine Learning*, 2003.

· Tetreault, J., et al. "A Report on the First Grammarly Shared Task for GEC." *BEA Workshop*, 2019.

· Zesch, T., et al. "Large-Scale Evaluation of Language Models for Spell Checking." *LREC*, 2022.

· Yang, Z., et al. "Universal Dependencies Parsing for Multi-Language Applications." *EMNLP*, 2018.

· Lopes, J. P., et al. "Text Normalization Techniques for User-Generated Content." *Information Processing & Management*, 2021.

· Heafield, K., et al. "Scalable Spell Checking with KenLM Language Models." *ACL Workshops*, 2018.