Change of Paradigms with AI-assisted Development

Edgardo Javier Trenti

Universidad Nacional de Salta, Argentina.
Facultad de Ciencias Exactas - Departamento de Informática.
http://di.unsa.edu.ar
jtrenti@di.unsa.edu.ar

Abstract. The proliferation of generative Artificial Intelligence has driven the emergence of *vibe coding*, a software engineering paradigm where developers manage high-level architecture rather than manually programming code. This paper presents a case study that examines the effectiveness of vibe coding in the end-to-end development of a fully automated news portal. Driven by prompt-mediated interactions with the Gemini Pro model, a full-stack architecture (FastAPI and React) was successfully deployed across a web server and a secure local AI server running a distilled Large Language Model (deepseek-r1). The system autonomously aggregates Argentine RSS feeds, performs semantic clustering, and generates neutral summaries. The results indicates a notable increase in productivity, alongside high systemic resilience and strict data sovereignty through the use of local open-source tools. Hardware-constrained local LLM responses introduced a moderated level of coherence errors and hallucinations. In contrast, the vibe coding methodology effectively shifted the developer's cognitive load toward architectural validation and system coherence. The study concludes that vibe coding is a highly viable first approach for rapid generation of relatively complex systems, laying the groundwork for future fine-tuning implementations based on triplet loss and other features to improve news writing.

Keywords: vibe coding, large language models, software engineering, generative ai, automated journalism, data sovereignty

Cambio de Paradigmas con el Desarrollo Asistido por IA

Abstract. La proliferación de la Inteligencia Artificial generativa ha impulsado el surgimiento del *vibe coding*, un paradigma de ingeniería de software donde los desarrolladores administran la arquitectura de alto nivel en lugar de código de programación manual. Este artículo presenta

un caso de estudio que examina la eficacia del vibe coding en el desarrollo integral de un portal de noticias completamente automatizado. Mediante interacciones guiadas por el modelo Gemini Pro, se implementó con éxito una arquitectura full-stack (FastAPI y React) distribuida entre un servidor web y un servidor de IA local que ejecuta un Modelo de Lenguaje Grande destilado (deepseek-r1). El sistema agrega de forma autónoma fuentes RSS de Argentina, realiza agrupamiento semántico y genera síntesis neutrales. Los resultados indican un notable incremento de productividad, junto con una alta resiliencia sistémica y una estricta soberanía de datos mediante el uso de herramientas de código abierto locales. Las respuestas LLM locales con limitaciones de hardware introdujeron un nivel moderado de errores de coherencia y alucinaciones. En contraste, la metodología de vibe coding desplazó efectivamente la carga cognitiva del desarrollador hacia la validación arquitectónica y la coherencia del sistema. El estudio concluye que el *vibe coding* es un primer enfoque muy viable para la generación rápida de sistemas relativamente complejos, sentando las bases para futuras implementaciones de ajuste fino basadas en *triplet loss* y otras características para mejorar la redacción de noticias.

Keywords: vibe coding, modelos grandes de lenguaje (LLM), ingeniería de software, inteligencia artificial generativa, periodismo automatizado, soberanía de datos

1 Introduction

The growing proliferation of large language models for code generation (Ge et al., 2025; Jiang et al., 2026), has fundamentally transformed the landscape of software engineering, enabling the rapid development of complex applications through human-guided orchestration. In this emerging model, the systems architect assumes the role of a context manager, iteratively refining outputs based on continuous feedback loops at each development stage. This paradigm shift has been conceptualized by Andrej Karpathy as vibe coding – a development methodology that emphasizes the communication of high-level design intentions rather than the manual implementation of source code.

Within the spectrum of emerging software development paradigms, several approaches have recently gained prominence, including *vibe coding*, *agentic coding* (Sapkota et al., 2025), and *Spec-Driven Development* (SDD) (Hassan et al., 2025; Piskala, 2026). These paradigms differ primarily in the degree of decision-making authority delegated to AI systems and in the level of formality applied to software specifications. In the vibe coding paradigm, a continuous interaction loop is maintained in which the human architect orchestrates code generation through generative AI, iteratively refining the generated artifacts to satisfy functional and integration requirements (Fawzy et al., 2025). Within this approach,

the architect retains primary responsibility for the strategic direction of software development. Conversely, agentic coding delegates a greater degree of autonomy to AI agents, allowing them to generate, review, and update project components with limited human intervention (Piskala, 2026; Sapkota et al., 2025). In this setting, the human role shifts toward supervising and validating the outcomes produced by autonomous agents. Recent research has also begun to establish formal frameworks for agentic coding (Piskala, 2026; Sabouri et al., 2025), many of which are grounded in SDD principles. These approaches emphasize the formalization of requirements, business rules, and use cases into explicit specification contracts before code generation begins (Hassan et al., 2025; Piskala, 2026). This evolution also reinforces a distinction between software engineering practices led by professional developers and application development initiated by end-users (Weber, 2025). Overall, while vibe coding emphasizes continuous human involvement during implementation, agentic coding prioritizes autonomous execution under human supervision, whereas SDD places formal software specifications at the center of the development process. Here we restrict ourselves to the vibe coding paradigm for the development of an automated news journal.

2 Methods

To evaluate the proposed approach, a fully automated news portal was developed. The system ingests RSS (*Really Simple Syndication*) feeds from newspapers across Argentina, including both national and regional media outlets, and automatically groups related articles into news events. For each event, it generates a consolidated news report while also analyzing the sentiment expressed in the individual articles. Event clustering, summarization, and sentiment analysis are performed using lightweight Large Language Models (LLMs).

2.1 System Architecture

The proposed system implements a fully automated online newspaper capable of collecting, processing, synthesizing, and publishing news articles without human intervention. Its architecture follows a modular container-based design, where each container encapsulates a specific responsibility within the news processing pipeline.

The system receives information from RSS feeds and publicly accessible web pages published by multiple Argentine newspapers. An ingestion and scraping component periodically retrieves new articles, removes irrelevant HTML content, and stores the resulting documents in a relational database. The same component orchestrates the AI processing pipeline by forwarding the collected articles to a local cognitive engine.

A scheduled AI Cognitive Engine executes a collection of lightweight Large Language Models (LLMs) running locally through Ollama. These models perform complementary natural language processing tasks, including semantic clustering of related news articles, summarization, sentiment analysis, and political

bias estimation. The generated metadata are subsequently persisted in the relational database, where they become available to the application layer.

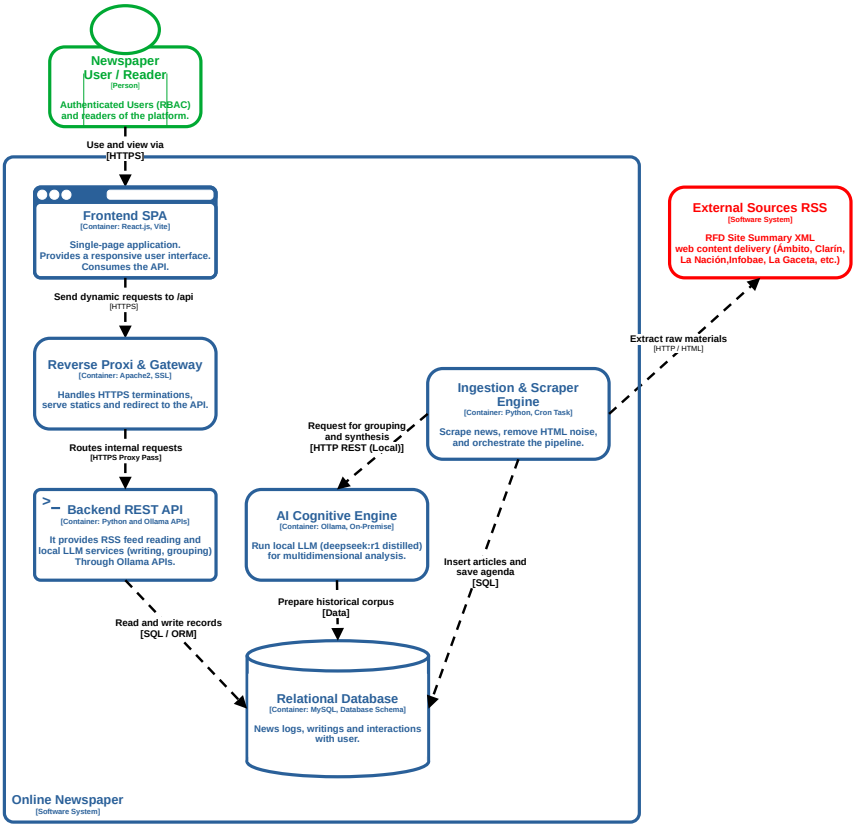

Fig. 1. Container diagram (C4 Model – Level 2) of the proposed automated news portal. The architecture is organized into six main containers responsible for user interaction, request routing, business logic, news ingestion, AI-driven cognitive processing, and data persistence. External news sources provide RSS feeds and web content, while locally deployed LLMs perform semantic clustering, summarization, sentiment analysis, and bias estimation before the processed information is delivered to end users.

The backend exposes a REST API responsible for authentication, business logic, and data access. It retrieves the processed news events and associated analytical information from the database and provides them to a single-page web application through an HTTPS gateway. The frontend presents the synthesized news stories, sentiment indicators, semantic tags, and historical information through a responsive user interface.

This modular organization decouples data acquisition, AI-based processing, persistence, and user interaction, allowing each subsystem to evolve indepen-

dently while facilitating the integration of new language models and analysis strategies.

2.2 Framework

The system was developed using Gemini Pro (Google AI Plus) through prompt-mediated interactions, following a classic structural pattern: role, context, task, and constraints. After evaluating various alternatives and trade-offs, a full-stack framework was adopted, comprising FastAPI (Python) for the backend and React (JavaScript/TypeScript) for the frontend. Data storage was initially implemented with SQLite to build a local Minimum Viable Product (MVP), which was later migrated to MySQL (MariaDB) on an Ubuntu Server managed via the Virtualmin control panel. It is worth noting that while the system architect has expertise in systems analysis, database design, and Python, their previous experience with React was highly limited. This highlights the impact of vibe coding in lowering the barrier to entry for adopting new technologies.

2.3 Web Server

The physical web server is equipped with an Intel Core i5 processor, 32 GB of RAM, and a 1 TB Solid State Drive (SSD). It runs Ubuntu Server and is remotely administered via Virtualmin. It features a symmetric 600 Mbps internet connection with a static public IP address. System-level administration is conducted over SSH using a non-standard port to enhance security. This configuration allows for the management of startup services within the corresponding virtual environment, including the SSH tunnel to the Artificial Intelligence server and the ASGI Uvicorn server for the FastAPI application. The main web service is initialized only after confirming the availability of the network and the AI tunnel:

```
[Unit]
After=network.target tunel-ia.service

[Service]
Environment="PATH=/home/automnewsjaiio/backend/venv/bin"
ExecStart=/home/automnewsjaiio/backend/venv/bin/uvicorn main:app
   -host 127.0.0.1 -port 8000
Restart=always
RestartSec=5
```

Given that the portal must be generated automatically and operate without human intervention, the `Restart=always` parameter ensures service resilience in the event of failure. As established in the `After` directive, this process depends on the connection service to the AI server.

2.4 AI Server

AI processing is handled by a server equipped with an Intel Core i9 processor, 64 GB of RAM, a 2 TB SSD, and an NVIDIA RTX 4070 GPU with 16 GB of VRAM, hosted at the *Departamento de Informática* of the *Universidad Nacional de Salta* (with restricted access via authorization).

The server locally runs the distilled deepseek-r1:latest model, based on Alibaba's Qwen3 architecture. This model features 8.19 billion parameters and utilizes Q4_K_M quantization (4-bit), offering an optimal balance between inference speed and accuracy. Occupying only 5.4 GB of memory, it is highly suitable for execution on the RTX 4070. The LLM, and API used for requests and responses is managed by Ollama (version 0.12.6).

Communication between the web server and the AI server is established via a system service that must be initialized prior to the dynamic generation of the website, which in turn requires network availability:

```
[Unit]
After=network.target

[Service]
Environment="AUTOSSH_GATETIME=0"
ExecStart=/usr/bin/autossh -M 0 -N -q
   -o "ServerAliveInterval 30"
   -o "ServerAliveCountMax 3" -o "ExitOnForwardFailure=yes"
   -L 10437:127.0.0.1:10437 userIA@xxx.xxx.xxx.xxx -p 9733
Restart=always
RestartSec=10
```

This service uses `autossh` to silently maintain the open tunnel. Periodic pings (ServerAliveInterval) are configured to check the AI server; if there is no response after three attempts, the process terminates, allowing `systemd` to detect the drop and restart the connection. The tunnel links a local port to the destination host, enabling FastAPI (on `localhost`) to communicate with the remote Ollama server through an encrypted channel. This architecture ensures that Ollama is not exposed to the public internet while providing fault tolerance.

To automate authentication for the encrypted SSH tunnel, asymmetric keys were generated and exchanged:

```
ssh-keygen -t ed25519 -C "tunel-ia automnews"
ssh-copy-id -p 9733 userIA@xxx.xxx.xxx.xxx
```

2.5 Procedure

The workflow is automated via a cron job that executes a Python module in the backend three times a day. This process collects the top five links from 19 different RSS feeds to build the front page. The LLM processes approximately

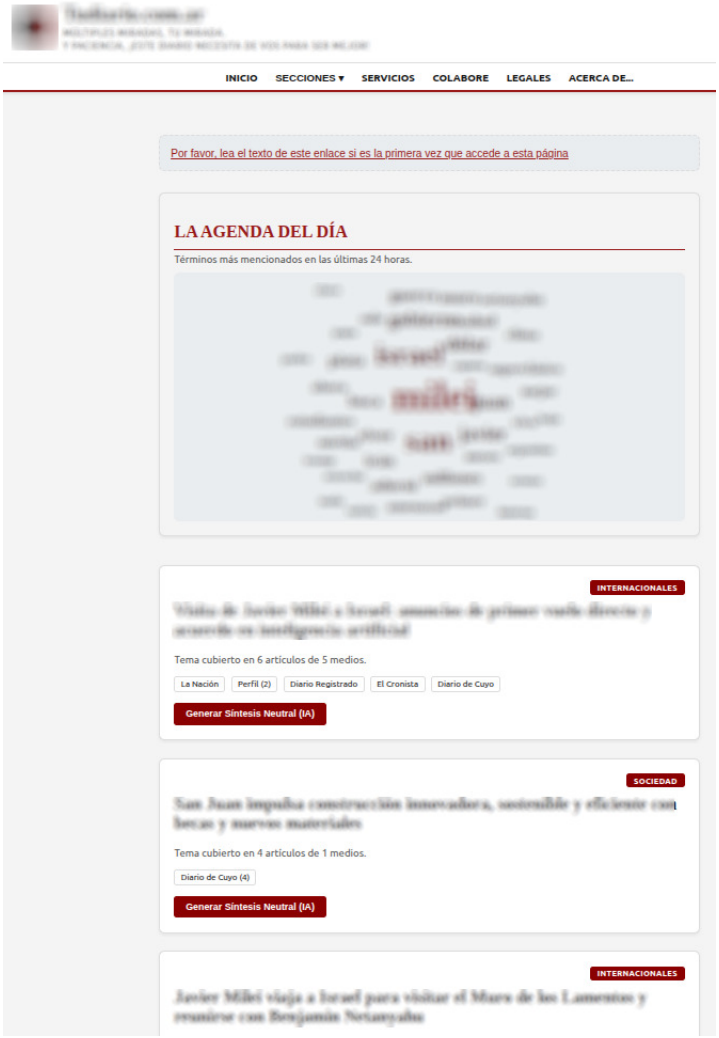

Fig. 2. Main interface of the automated news portal. The layout features a word cloud representing the daily agenda, followed by the news cards clustered by topic. The interface employs a responsive design, ensuring optimal rendering and user experience across mobile devices.

80 articles during each execution of the Python module to assess their bias and positivity levels, as well as to determine their corresponding thematic category.

For news clustering, the SBERT (Sentence-Transformers) package is used locally in conjunction with a community detection module. Articles whose titles exhibit a semantic similarity greater than 0.6 in their embeddings are grouped as the same event, and a new representative title is synthesized from the originals.

Subsequently, a consolidated summary is generated using between two to five sources from the same cluster, aiming to average and balance the inherent biases. The prompt sent to the LLM enforces strict constraints: maintain a neutral tone, rely exclusively on the information provided in the source texts (preventing hallucinations or outside assumptions), and adhere to specific formatting rules. The result is dynamically published via an auto-generated slug page.

The user interface was iteratively refined during development based on the system architect's user experience principles. Additionally, capitalizing on the processing of over 80 daily articles, a word cloud was implemented to visualize the *"issues of the day."* Notably, when faced with library installation dependency conflicts, vibe coding was utilized to program a custom local script to render the word cloud. Figure 2 shows the final interface of the automated portal's front page.

Upon accessing the main interface, first-time users are presented with a notice that, among other platform features, outlines its legal framework. Specifically, the disclaimer states: "Rights and Attribution: All news items aggregated herein remain the intellectual property of their respective publishers and journalists. AutomatednewsJAIIO utilizes excerpts of these works under the Right to Quote (Article 10 of the Argentine Republic's Intellectual Property Law 11.723) for strictly informational and comparative analysis purposes ('multiple perspectives'). The original source is consistently cited and linked to foster transparency and drive traffic to the original authors."

3 Results

The implementation of a local open source LLM presented the primary advantage of eliminating dependencies on commercial APIs (such as the Gemini API), whose free tiers are rapidly exceeded by the query volume of this automated system. However, constraints regarding access to high-end GPUs limited the project to utilizing a smaller-scale local model. Consequently, title generation and summary drafting incurred coherence errors and hallucinations at a rate that is currently unacceptable for a commercial production environment. Figure 3 illustrates some of the anomalies detected during development. Currently, a monitoring dashboard is under development to statistically quantify the error rate in text generation.

Despite these limitations, qualitative evaluation indicates that the majority of published articles achieve an adequate presentation of the facts, offering neutral summaries. Notably, no systematic biases introduced by the LLM have been observed; if an inherent bias pattern exists within the model, it remains imperceptible through direct human evaluation. Figure 4 showcases a representative example of a concise summary synthesized from two divergent sources.

Over an operational period of approximately 45 days, the system processed an average volume of 240 articles daily, equating to 80 articles for each of the three daily executions of the dynamic generation module. The average execution time

Fig. 3. Examples of common errors detected during the evaluation of the lightweight language model. Instances of blatant hallucinations, as well as drafting or grammatical concordance failures, are distinguished.

for the complete pipeline—comprising RSS feed ingestion, semantic clustering via SBERT, and LLM inference—was established at approximately 40 minutes.

The network and communication infrastructure demonstrated a superlative level of stability. The service experienced only two interruptions, both attributable to physical power outages (one at the University facilities and the other at the web server's location). In both scenarios, the system recovered fully without manual intervention, thanks to the automatic restart policies configured on the servers in the event of power loss.

The vibe coding paradigm significantly accelerated the development of the system's core. The configuration of the service architecture, inter-server communication, and automated recovery strategies were designed and implemented under the exclusive guidance of the Gemini Pro model. This highlights the paradigm's capability to bridge domain-specific knowledge gaps, enabling a single developer so successfully architect and deploy the complete system.

Based on this experience, a critical methodological recommendation is formulated: discarding the "copy-paste" approach often suggested to accelerate implementation. The manual transcription of the LLM-generated code into an Integrated Development Environment (in this case, VS Code) facilitates comprehensive review, allowing the developer to identify and correct erroneous assumptions in the AI's architectural design. Furthermore, efficient management of the context window (token limit) is imperative. During extended interactions, it was observed that the LLM tends to lose track of developmental guidelines established

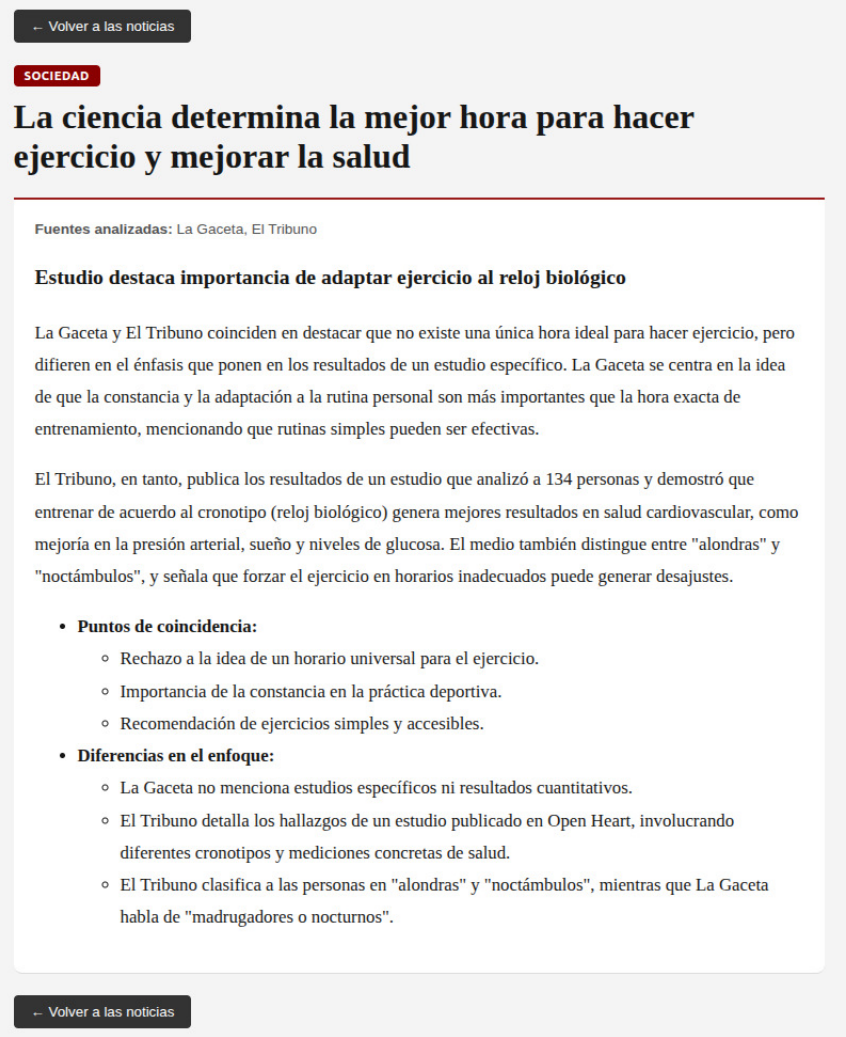

Fig. 4. Example of a concise summary generated from a single news event reported by two different journalistic sources. The newspaper El Tribuno (Salta) provides the article titled "Jet lag social: cuál es la mejor hora para hacer ejercicio según tu reloj biológico y mejorar tu salud," while La Gaceta (Tucumán) contributes the piece "Cuál es la mejor hora para hacer ejercicio: por qué el horario influye en la salud, según la ciencia."

in the early stages of the process. To mitigate this, it is highly recommended to leverage the project customization tools or persistent system instructions currently offered by most models.

Finally, it should be noted that this study is limited to an experience in pure vibe coding. Naturally, the ecosystem is evolving toward new tools that ex-

pand software production capabilities, such as agentic coding. From the outset, these systems display a superior level of integration through the use of skills, autonomous planning, automated testing, and database management within a unified workspace to which agents have full access. This approach prevents structural contradictions and the introduction of cross-contamination errors typical of fragmented conversational interactions.

4 Discussion

The system, currently in an iterative phase, is developed through a staged architecture defined by the project leader. Implementation details, component integration strategies, and the source code (both backend and frontend) were generated using the Gemini Pro model within a vibe-coding workflow. This development-time model is distinct from the AI models deployed in the operational system, where a locally hosted distilled DeepSeek-R1 model performs sentiment analysis, bias estimation, and abstractive summarization, while semantic clustering relies on multilingual sentence embeddings. This inherent characteristic of vibe coding facilitates a shift in the developer's cognitive load: by offloading concerns regarding detailed syntax and the mastery of diverse libraries, the professional can focus their faculties on architectural validation, system coherence, and the mitigation of inconsistencies generated by the model.

Although the generated source code was continuously reviewed by the project leader during development, no formal static code analysis has yet been conducted. Future work will incorporate software engineering quality metrics, including cyclomatic complexity, maintainability index, code duplication, coupling, cohesion, and technical debt estimation. These measurements will enable a quantitative assessment of the impact of vibe coding on software quality and long-term maintainability.

A critical consideration in integrating AI services is data protection. The proposed design, grounded in locally executed open-source tools, ensures data sovereignty and the unaltered preservation of information. Although this case study utilizes public news sources, the methodology allows for the establishment of protocols that prevent the exposure of sensitive data to third-party services. This architectural benefit is vital for institutional projects requiring strict control over data privacy and integrity.

In its operational state, the platform exhibits a high availability rate and a resilient fault recovery schema. The system has proven transparent to the user during incidents such as AI server overloads, network fluctuations, or power outages. A principle of graceful degradation was implemented: in the event of AI server failures, the site remains operational via integrated error messages; only a power loss at the primary web server results in a service interruption external to the system.

Beyond availability, future evaluations will assess compliance with both functional and non-functional requirements. Particular attention will be devoted to performance, response time, scalability, maintainability, fault tolerance, and us-

ability, complementing the software quality metrics obtained through static analysis.

Future work will also explore alternative lightweight language models for the news summarization task. The current implementation employs a distilled version of DeepSeek-R1 as a general-purpose reasoning model capable of performing multiple NLP tasks. Although this simplifies deployment, reasoning-oriented models are not necessarily the most computationally efficient solution for abstractive summarization. Consequently, future versions of the platform will evaluate smaller instruction-tuned summarization models, including compact variants of Llama, Qwen, T5, and BART, in order to reduce inference latency and computational cost while preserving summary quality.

We will also include the implementation of a Role-Based Access Control (RBAC) system to segment readers, collaborators, and administrators. Readers will have access to customization options to request summaries with specific sources, and the interface will log collaborator interactions to compile a structured dataset. Collaborators will act as human-in-the-loop validators, assessing bias accuracy and the relevance of semantic clustering. This dataset will subsequently be used to train a custom AI model aimed at optimizing and refining the semantic precision of news headline clustering, effectively establishing a Reinforcement Learning from Human Feedback (RLHF) pipeline.

To date, the system has processed a corpus of 10,570 articles. This growing database will facilitate the transition to a Retrieval-Augmented Generation (RAG) architecture, grounding the drafting process in historical knowledge to reduce hallucinations – a goal that will require scaling the computing infrastructure. Furthermore, there are plans to explore agentic coding using environments such as Google's Antigravity, Cursor, or Claude Code, and also evaluate the impact of Specification-Driven Development (SDD). In addition, controlled experiments comparing vibe coding and agentic coding will be conducted using the proposed software quality metrics and requirement compliance indicators to objectively evaluate their impact on development productivity and software quality.

In conclusion, the experience of employing vibe coding for the development of a full-cycle application has proven highly satisfactory, suggesting a substantial increase in development productivity. Beyond the technical success of building an autonomous portal, this process has allowed for the accelerated assimilation of complex network concepts and AI architectures, demonstrating that this paradigm not only optimizes development cycles but also serves as a catalyst for advanced technical learning.

Acknowledgments. Trabajo realizado con soporte del Departamento de Informática. Facultad de Ciencias Exactas. Universidad Nacional de Salta.

Disclosure of Interests. This text is the product of a human translation, with subsequent correction assistance from Gemini Pro.

References

Fawzy, A., Tahir, A., & Blincoe, K. (2025). Vibe coding in practice: Motivations, challenges, and a future outlook – a grey literature review. https://arxiv.org/abs/2510.00328

Ge, Y., Mei, L., Duan, Z., Li, T., Zheng, Y., Wang, Y., Wang, L., Yao, J., Liu, T., Cai, Y., et al. (2025). A survey of vibe coding with large language models. *arXiv preprint arXiv:2510.12399*.

Hassan, A. E., Li, H., Lin, D., Adams, B., Chen, T.-H., Kashiwa, Y., & Qiu, D. (2025). Agentic software engineering: Foundational pillars and a research roadmap. *arXiv preprint arXiv:2509.06216*.

Jiang, J., Wang, F., Shen, J., Kim, S., & Kim, S. (2026). A survey on large language models for code generation. *ACM Transactions on Software Engineering and Methodology*, *35*(2), 1–72.

Piskala, D. B. (2026). Spec-driven development: From code to contract in the age of ai coding assistants. *arXiv preprint arXiv:2602.00180*.

Sabouri, S., Eibl, P., Zhou, X., Ziyadi, M., Medvidovic, N., Lindemann, L., & Chattopadhyay, S. (2025). Trust dynamics in ai-assisted development: Definitions, factors, and implications. *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, 1678–1690. https://doi.org/10.1109/ICSE55347.2025.00199

Sapkota, R., Roumeliotis, K. I., & Karkee, M. (2025). Vibe coding vs. agentic coding: Fundamentals and practical implications of agentic ai. *ArXiv*, *abs/2505.19443*. https://api.semanticscholar.org/CorpusID:278905853

Weber, I. (2025). Feasibility of ai-assisted programming for end-user development. *arXiv preprint arXiv:2512.05666*.