Revista Eletrônica de Iniciação Científica em Computação, 2026, 24:1
ISSN: 3085-8461 • doi: 10.5753/reic.2026.6680

Research Paper

Tag-Driven File Management System

Luiz Fernando Lemos [Federal Center for Technological Education Celso Suckow da Fonseca | *luizfernandosilvalemos845@gmail.com*]
Diego Castro [Federal Center for Technological Education Celso Suckow da Fonseca |*diego.casto@cefet-rj.br*]
Carlos Eduardo Pantoja [Federal Center for Technological Education Celso Suckow da Fonseca |*pantoja@cefet-rj.br*]
Bruno Freitas [Federal Center for Technological Education Celso Suckow da Fonseca|*bruno.freitas@cefet-rj.br*]

R. Miguel Ângelo, 96 - Maria da Graça, Rio de Janeiro - RJ, 20785-220

Abstract. The search for files is an essential task in the daily lives of computer users. Still, conventional systems can be inefficient, especially when dealing with large amounts of files, leading to wasted time and reduced productivity. This work proposes a Tag-Driven File Management System (TFMS) as an alternative to optimize file organization and retrieval, enabling contextual searches and dynamic categorization. The application was designed with an emphasis on performance, created as a fast and secure desktop application that utilizes a database to store the tags associated with the files, while a caching system enhances directory loading efficiency. Additionally, artificial intelligence models were integrated to suggest tags automatically, enhancing the user experience. The results show that TFMS demonstrated superior efficiency compared to traditional methods, significantly reducing file search time. The comparison with conventional systems validates the effectiveness of the tag-based approach, positioning TFMS as a promising solution for file management.

Keywords: File Management, Tag-based Systems, File Organization, Information Retrieval, Efficiency

Received: 13 October 2025 • **Accepted:** 20 February 2026 • **Published:** 20 March 2026

1 Introduction

Every day, the quantity of digital information is expanding exponentially. Some experts have stated that the organization and accessibility of files have become more challenging due to this increase (SINGH *et al.*, 2007). Typically, files are organized in a hierarchical directory structure; however, this approach could become unsatisfactory as the volume of documents expands. Complex structures, composed of several levels of directories, make navigation difficult, slower, and more susceptible to errors. Additionally, this organization can potentially promote file duplication, which results in inconsistent classification and complicates locating specific documents. Consequently, the efficacy of file searches may be compromised, despite the application of indexers, such as those integrated into operating system search engines (LIU *et al.*, 2018).

According to some studies, office employees claim to spend a significant portion of their time finding files within disordered systems and can spend hours on these activities (Dinneen and Julien, 2020). Moreover, studies indicate that frustration from difficulty locating desired files can result in creating new files containing identical content, worsening disorganization, duplicating data, and increasing storage requirements, which aggravates the file search process and generates a "snowball" effect. As a result, numerous companies allocate significant financial resources to File Management Systems (FMS) and document storage systems (JACKSON and SMITH, 2012).

However, there are more efficient and low-cost alternatives for organizing files that can overcome the limitations imposed by traditional hierarchical structures. One potential solution is the implementation of structured metadata, which includes descriptive qualities associated with files, such as creation date, author, category, or document type. Search engines can use this information to filter search results. In addition to this option, tags are employed to designate keywords to files to describe their content, context, purpose, or any other pertinent characteristic. In contrast to conventional folder organization, which requires a rigorous and hierarchical structure, the utilization of tags enables the same file to be associated with multiple categories simultaneously, including theme, project, and purpose ALBADRI *et al.* (2016); JACKSON and SMITH (2012). This method is more content-centric, dynamic, and flexible, enabling the retrieval of files through contextual searches based on more natural criteria, without requiring the user to remember the precise path where the file is stored.

Despite the exploration of tag-based systems and intelligent file organization tools in previous research, and their integration into many commercial platforms, significant limitations persist. Many current options are proprietary, rely on cloud infrastructure, or are designed for enterprise-level document management, which restricts user control, transparency, and flexibility. Moreover, automated tagging systems often operate as black-box models, providing little transparency about how tags are created or how semantic connections are established. Users often integrate manual organizing methods with external tools, suggesting that existing systems inadequately meet cognitive, contextual, and personalization requirements in routine file management. Consequently, despite technological advancements, a disparity persists between existing solutions and a flexible, user-centric, transparent, and AI-enhanced file management approach (Dinneen and Julien, 2020; JACKSON and SMITH, 2012).

In response to these limitations, this work proposes the development of a Tag-Driven File Management System (TFMS) to improve the efficiency of organizing, retrieving, and maintaining files. The tags are intentionally designated by the user, reflecting their interpretation and need for organization, which is different from the metadata proposal that

is usually generated by the operating system. This model is a paradigm shift in comparison to traditional methods, as it facilitates a search experience that is more adaptable, precise, and agile in response to the requirements of users. In addition, the tool's integration with artificial intelligence (AI) aims to improve the user experience by automatically suggesting tags based on the semantic content of the files. This will facilitate the organization process and minimize the need for manual intervention.

The remainder of this paper is organized as follows: Section 2 presents a brief review of the literature with the intention of finding and describing related works. Section 3 describes the development and features of TFMS. Section 4 presents the evaluation results. Section 5 concludes with future directions.

2 Rapid Review

A rapid review (DOBBINS, 2017) was conducted to identify relevant literature. While this study diverged from the conventional systematic review approach by not formulating explicit research questions, it nonetheless utilized a structured and transparent methodology to guide the search, screening, and selection phases. The primary objective was to ensure scientific validity and replicability in the identification of pertinent studies, concurrently preserving the efficiency inherent in rapid review approaches.The investigation was conducted from May to July 2024 and intended to identify relevant research in file organization within operating systems. The search string was executed in the Scopus research database (https://www.scopus.com) as recommended by Kitchenham (KITCHENHAM, 2004), returning 554 results. This work sought to extract the characteristics and techniques found in each of the related works found.

- The inclusion criteria specified that:
  - the paper must be related to applications for tag-based file management;
  - the paper should constitute a primary or secondary study; and
  - the paper must address at least one of the research issues.
- The exclusion criteria indicated that:
  - book chapters and conference invites should be removed;
  - papers lacking full access should be discarded; and
  - research outside the domains of computer science or engineering should be rejected.
- The implementation of these criteria and filters significantly diminished the quantity of papers reviewed, decreasing from 121 post-title analysis to 32 following abstract screening and ultimately to merely 8 after comprehensive text examination. The papers selected for review can be identified in Table 2.

The search string was generated utilizing the PICOC (population, intervention, comparison, outcome, and context) framework (BRUZZA *et al*., 2017). Four of the five components were utilized to fragment the string, with the objective of enhancing the efficacy of the results achieved. Additionally, certain words were terminated with the character '*' to encompass terms that are derived from the original word.

Upon identifying the synonyms, the search strings for the study may be formulated. The string was produced by aggregating keywords from the same domain using the logical operator "OR" and consolidating distinct domains with the logical operator "AND". The components of the PICOC and the employed search string are specified in Table 1.

2.1 Analysis of findings

The examined systems provide several methods to enhance organization and performance in operating systems, emphasizing efficient file retrieval and access. TagFS enables users to attach tags to files, hence enhancing their retrieval and organization. These tags are retained in Non-Volatile Memory (NVM), together with the files' information, ensuring their accessibility post-reboots or system failures. To enhance file search efficiency, TagFS employs a B+ Tree, wherein the leaves consist of tags accompanied with metadata and file identifiers (BEZEK, 2011). A comparable paradigm is employed in FindFS, which incorporates a dynamic visualization layer based on tags into an existing hierarchical file system, enabling users to filter and retrieve files in an organized and efficient manner, with directories having symbolic links to files and real-time updated visualizations. The system employs a caching mechanism to store frequently accessed tags and files, thereby enhancing performance (CHOU, 2015).

Certain systems examined employ methods primarily aimed at optimizing I/O operations. Batch File Operations (BFO) employs a batch operations concept to enhance access to numerous tiny files. It achieves this by segregating the processing of metadata and files, hence enhancing the efficiency of I/O operations (YANG *et al*., 2020). A separate file system examined, known as NoseFS, utilizes a global hash table for the organization and retrieval of metadata. An index tree is utilized to execute read and write operations efficiently, while a lightweight journaling system is integrated to maintain system consistency post-failure. In contrast to more substantial systems, lightweight journaling captures solely the essential information for data recovery, ensuring that system performance remains uncompromised, while write operations are tuned to minimize I/O overhead (YANG *et al*., 2018). In contrast, Otter (LIU *et al*., 2019) employs a strategy centered on metadata localization, providing an efficient indexing structure both in memory and on disk, while prohibiting the utilization of tags.

Otter suggests a metadata organizing method centered on metadata localization, utilizing a constant-complexity indexing framework for both memory and disk, hence facilitating swift file retrieval. The I/O optimization method is evident in systems utilizing the FUSE framework, which is intended to aid in the creation of user-space file systems, as outlined by (VANGOOR *et al*., 2019). FUSE facilitates effective communication between user space and the kernel, enhancing flexibility in the creation of bespoke file systems. DEFUSE, an iteration of FUSE, use tags to classify files within a virtual directory, irrespective of their physical disk locations. This system utilizes a caching mechanism to preserve frequently accessed tags and files, hence enhancing speed (LEMBKE *et al*., 2022). XFUSE is a FUSE-based system that employs a tag-based technique for file organization, enabling users to apply multiple tags to files and improving efficient file re-

Population: "operational system*" OR "Operating system*" OR "windows" OR "linux" OR "OS" OR "mac*" OR "unix" OR "debian" OR "ubuntu"
Intervention: "file management" OR "file organization"
Comparison: Not applicable
Outcome: "application*" OR "program*" OR "software*" OR "system*"
Context: "eficiency" OR "fast search" OR "user satisfaction" OR "performance" OR "productivity"OR "efficacy" OR "facility".
TITLE-ABS-KEY (("operational system"OR "Operating system*"OR "windows"OR "linux"OR "OS"OR "mac*"OR "unix"OR "debian"OR "ubuntu") AND ("file management" OR "file organization") AND ("application*"OR "program*"OR "software*"OR "system*") AND ("eficiency"OR "fast search" OR "user satisfaction"OR "performance"OR "productivity" OR "efficacy"OR "facility")) AND PUBYEAR >2014 AND PUBYEAR <2025 AND (LIMIT-TO (SUBJAREA , "ENGI") OR LIMIT-TO (SUBJAREA , "COMP"))*

Table 1. String de busca

Number	Title	Reference
1	An Efficient and Flexible Metadata Management Layer for Local File Systems	(LIU *et al*., 2019)
2	Performance and Resource Utilization of Fuse User-Space File Systems	(VANGOOR *et al*., 2019)
3	Batch-file Operations to Optimize Massive Files Accessing	(YANG *et al*., 2020)
4	DEFUSE An Interface for Fast and Correct User Space File System Access	(LEMBKE *et al*., 2022)
5	A Highly Non-Volatile Memory Scalable and Efficient File System	(YANG *et al*., 2018)
6	F2FS: A New File System for Flash Storage	(LEE *et al*., 2015)
7	XFUSE An infrastructure for running filesystem services in user space	(HUAI *et al*., 2021)
8	A high performance file system for non-volatile main memory	(OU *et al*., 2016)

Table 2. Selected papers.

trieval using tag-based searches (HUAI *et al*., 2021). F2FS is a file system specifically engineered for flash memory devices, such as SSDs. It employs a system that differentiates between "hot" data, marked by frequent access or alteration, and "cold" data, comprising files that are infrequently accessed. This distinction enhances the efficient arrangement of writing and reading processes, reducing fragmentation and degradation of flash memory (LEE *et al*., 2015).

Comparable frameworks, such as DEFUSE, prioritize user-space interaction, offering an interface that improves file descriptor acquisition and retention, while also enabling user-space paging. XFUSE provides an infrastructure that allows file system services to function in user space, enabling directory overlays and the mounting of distinct file systems within containers, hence improving flexibility and scalability across diverse storage environments. F2FS is a file system explicitly engineered for flash memory devices, including SSDs. It utilizes a methodology that differentiates between "hot" data, characterized by frequent access or modification, and "cold" data, including files that are seldom utilized. This differentiation facilitates the effective structuring of write and read operations, minimizing fragmentation and deterioration of flash memory (LEE *et al*., 2015).

The examination of the tools revealed a variety of distinct features. Table 3 presents a thorough comparative analysis of the tools under consideration. Each table entry corresponds to a specific attribute, and an "X" indicates the presence of that attribute in the respective tool. This comparative analysis highlights the absence of a single tool that includes all the identified features. Therefore, the proposed TFMS is justified because it combines the complete range of features found in the tools that were examined, resulting in a single, integrated application.

3 Tag-Driven File Management System

The strategies identified in the examined file systems inspire the system proposal presented in this study. As a result, the proposal used similar techniques to those mentioned earlier, leading to advantages like robustness, adaptability, and high efficiency, which are essential for handling large amounts of data.

The selection of technologies for execution reflects an effort of efficiency and security. The Tauri framework facilitates the creation of flexible desktop applications utilizing the Rust programming language for the backend and web technologies, such as React, for the frontend (Sinha *et al*., 2025). Its modular architecture enables integration with Artificial Intelligence (AI), while its simple interaction with operating

	TagFS	FindFS	BFO	NoseFS	Otter	DEFUSE	XFUSE	F2FS	TFMS
Use of Tags	X	X							X
B+ Trees	X								X
Refreshing Views		X							X
Batch Operations			X						X
Persistence in NVM	X			X	X				X
Global Hash Table				X					X
Metadata Locality					X				X
Hot/Cold Data Separation								X	X
Userspace Implementation		X			X	X	X		X

Table 3. Comparison of application features.

system features permits sophisticated file manipulation. Rust, a high-performance language with exact memory management and inherent support for safe concurrency, encompasses the essential elements for applications requiring robustness and reliability, such as cache and file management systems.

The application cycle starts with cache initialization and database connection, followed by interface rendering. The user navigates the system directories, locates specific files by name or tags, and manages the relationships between tags and files. The backend manages requests and modifies the database as required. The interface subsequently presents the results. The interface was segmented into three primary components to enhance maintenance and scalability. The navigation bar component contains folder navigation and a search bar equipped with advanced filters. The main area exhibits the files and directories together with their corresponding tags. Context menus facilitate rapid actions, including renaming or deleting files, assigning tags via modals, or creating folders. This division into discrete elements enhances the readability and maintainability of the code, facilitating the incorporation of future enhancements.

The application architecture consists of four primary components: the user interface (frontend), the application core (backend), the data persistence layer (database), and supplementary modules, including caching and AI integration. The SQLite database is assigned the persistent storage of information pertaining to files, tags, and their associations, maintaining a hierarchical structure and referential integrity among tables.

The backend oversees file operations, interfaces with the database, and incorporates AI libraries. The structure, along with that of the frontend, is depicted in Figure 1. The application modules are systematically arranged into directories based on their functions, including filesystem, search, cache, database, and AI, with each directory housing unique implementations of its respective features. These modules facilitate activities such as file listing and deletion, detection of mounted volumes (e.g., external hard drives or network drives), and cache system management. The cache serves as an intermediary layer between the file system and the backend, maintaining a persistent representation of directories in a B+ tree structure. This facilitates efficient loading, even in contexts with substantial file quantities. The cache is serialized to disk, verified, and updated at regular intervals. This layer not only enhances performance substantially but also serves as a vital integration point among the directory explorer, search engine, and volume manager.

The TFMS has two complementary search methodologies: nominal search and tag search. Nominal search enables the identification of files by their names, accommodating typographical errors with a fuzzy matching algorithm that analyzes the data set and categorizes the outcomes according to a similarity index. This method can discover partial matches and rank the files based on the proximity of the search phrase to the real file names, providing a fault-tolerant and more intuitive search experience for the user. Tag search functions according to the keywords previously allocated to the files. Each file may encompass one or several tags that semantically characterize it, facilitating more efficient thematic or contextual searches for the user. This search method relies on the words included inside the database rather than the file names, enabling the identification of content even when the file names lack descriptiveness. These two search engines operate cohesively, enhancing information retrieval flexibility and markedly boosting the system's navigability and usability, particularly in contexts with substantial amounts of unstructured data. The TFMS interface are illustrated in Figure 2.

The integration of AI technologies into TFMS is executed in a modular and autonomous design, offering flexibility and facilitating system maintenance. The system sends file metadata, such as name, extension, path, and previously related tags, to an external API called Hugging Face (Jain, 2022), rather than depending exclusively on a predefined set of tags or manual operator intervention. This API allows a zero-shot classification model to automatically suggest new tags based on the meaning of the provided metadata, without needing any previous training. This enables the system to dynamically adjust to the document content, suggesting pertinent and precise tags based on context, a capability not attained by traditional file management techniques.

The AI-generated tag suggestions are subsequently analyzed and categorized based on their relevancy, displaying the most suitable options to the user. This approach enhances file organization on disk and substantially diminishes the manual effort needed for categorization and ongoing file updates, which is a common difficulty in conventional file manage-

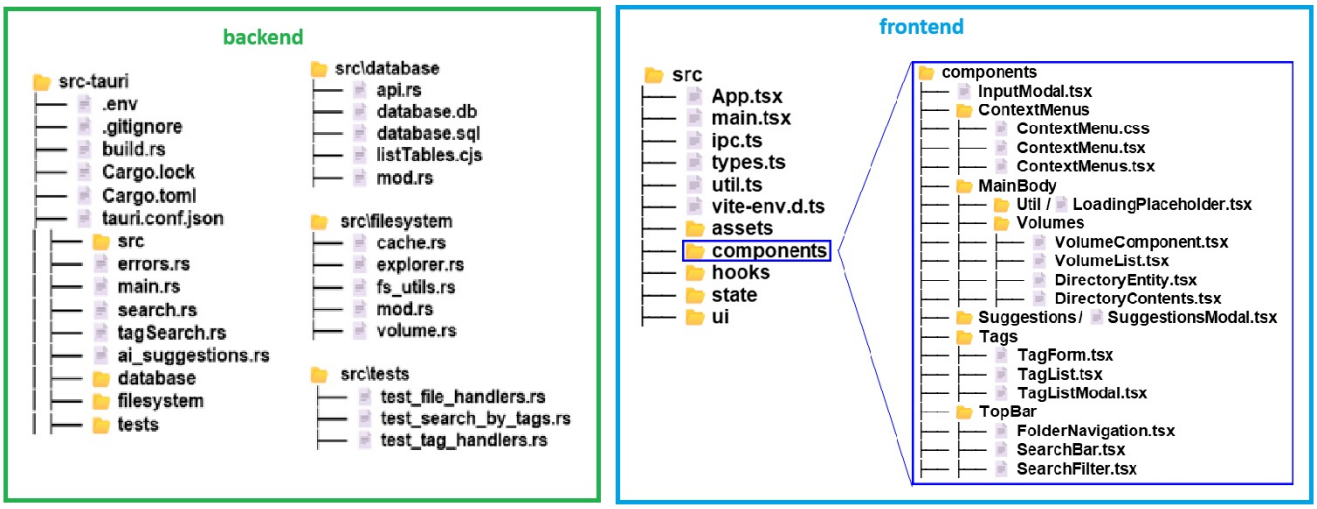

Figure 1. Application folder structure.

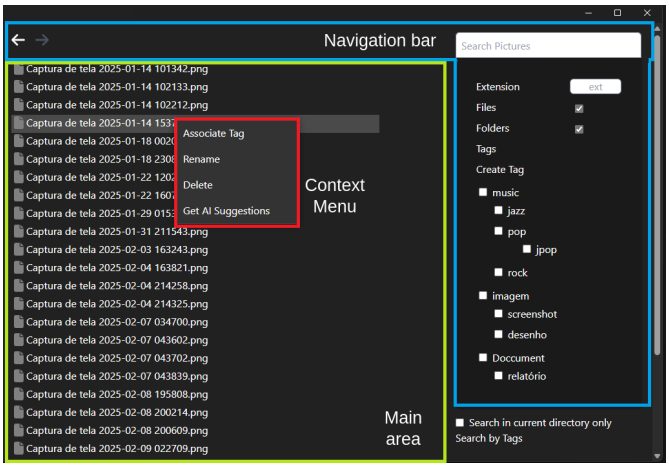

Figure 2. Software interface.

ment systems. Moreover, the implementation of this AI layer as an independent module, adhering to the principles of low coupling and high cohesion, facilitates the rapid expansion and modification of the system without impacting the other software layers. This modular model preserves the system's original structure while facilitating the integration of future technology advancements at a minimal adaptation cost.

4 Evaluation

The initial performance studies of the TFMS were carried out by comparative analyses with the Windows File Explorer, concentrating on file search duration. The assessments were conducted in two experimental scenarios: one initiating the search from the system root directory and the other from the directory containing the target file. Each type of search (nominal and by tags) was repeated three times for both circumstances. The experiment's findings demonstrated a substantial enhancement in system performance compared to the conventional Windows Explorer strategy. The notional search exhibited, on average, a response time 6.69 times faster in TFMS. The tag-based search has shown a markedly greater improvement, proving to be nearly 16,800 times more effective in file retrieval. This enhanced performance results from the SQLite database query, which, when files are appropriately organized by tags, navigates a significantly narrower dataset compared to a comprehensive file system scan.

In addition to the assessment utilizing the conventional search method via the operating system, an attempt was made to evaluate the proposed tool against the most closely related works to this proposal; consequently, the tools FindFS (CHOU, 2015) and TagFS (BEZEK, 2011) were chosen. Access to the application executables or source code of the tools was not possible, necessitating that the evaluations be conducted only on the information provided in the papers detailing the tools. Nevertheless, the article TagFS omitted to provide evaluation results, leading to its exclusion from the assessment.

The papers' results indicated that TFMS exhibited inferior performance compared to FindFS, which was anticipated due to the differing characteristics of their architectures. FindFS functions directly within the conventional hierarchical structure employed in Linux, resulting in overhead when intercepting actions at the file system level. Conversely, TFMS employs a database-centric architecture for file and metadata management, providing enhanced flexibility and control; however, this may adversely affect raw performance in basic operations, particularly when contrasted with tools tuned for system-level efficiency. In the initial test scenario, FindFS exhibits query times between 3.32 and 10 µs for file searches from the machine's root directory, but tag-based searches in TFMS reveal response times ranging from 0.48 to 2.23 ms. It is important to recognize that the search methodologies differ, as TFMS does not require managing result set updates, a factor that imposes considerable overhead on FindFS operations. Ultimately, it is important to note that the evaluations were conducted based on reports in the publications, and the assessments occurred on computers with varying settings, which may jeopardize the validity of the comparison between FindFs and TFMS.

While the experimental findings indicate consistent performance enhancements, certain facets of the evaluation warrant contextualization. The experiments were executed on the author's personal computer to ensure a controlled and stable execution environment throughout all scenarios. All tests were conducted under identical hardware configurations, operating system conditions, and datasets, thereby guaranteeing

internal consistency and facilitating comparability between TFMS and the other programs The principal metric assessed was file search response time, given that this metric directly reflects the system's fundamental objective of enhancing retrieval efficiency.

The exact number of files in each directory wasn't formally tracked during the experiment. This was because the study focused on functional and comparative validation, not detailed performance testing. Therefore, even though all runs used the same dataset and file structure, the lack of precise dataset details could potentially affect the ability to reproduce the results exactly. Future research will use controlled datasets, with specific file sizes and standardized testing methods, to improve external validity and generalizability.

5 Conclusion

This paper proposes a Tag-Driven File Management System (TFMS), an effective solution for file organization and retrieval, prioritizing performance and adaptability. TFMS presents a distinct methodology for tag utilization, facilitating file classification and search in a more dynamic and contextual manner. The performance evaluation of TFMS versus Windows Explorer demonstrated a notable enhancement in search velocity, particularly in the tag search approach, which was much quicker in scenarios with extensive data files. The system distinguished itself through its integration with Artificial Intelligence, which autonomously recommends tags, enhancing the organizational process.

Despite TFMS's worse performance compared to FindFS, attributable to its database-centric architecture that incurs greater overhead from file and metadata management via queries, this modular design offers considerable benefits in flexibility and scalability. Utilizing a database facilitates enhanced control over data organization and simplifies the incorporation of additional functionalities, such as AI integration for automatic tag suggestions. This distinguishing characteristic enhances file management and optimizes the user experience, rendering the procedure more efficient and intuitive. The research validated TFMS as an efficient solution for file management, emphasizing its capacity to manage substantial data quantities in a more systematic and customized manner.

Future research may investigate the deployment of AI algorithms to enhance tag suggestions and provide ways to optimize performance in basic tasks, such as nominal searches. Implementing a more sophisticated caching mechanism could enhance performance in situations involving substantial file quantities, rendering TFMS an even more resilient and efficient solution.

References

ALBADRI, N., Watson, R., and Dekeyser, S. (2016). Treetags: bringing tags to the hierarchical file system. In *Proceedings of the Australasian Computer Science Week Multiconference*, pages 1–10.

BEZEK, S. (2011). Tagfs: A simple tag-based filesystem.

BRUZZA, M., Cabrera, A., and Tupia, M. (2017). Survey of the state of art based on picoc about the use of artificial intelligence tools and expert systems to manage and generate tourist packages. In *2017 International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions)(ICTUS)*, pages 290–296. IEEE.

CHOU, J. (2015). *FindFS: adding tag-based views to a hierarchical filesystem*. PhD thesis, University of British Columbia.

Dinneen, J. D. and Julien, C.-A. (2020). The ubiquitous digital file: A review of file management research. *Journal of the Association for Information Science and Technology*, 71(1):E1–E32.

DOBBINS, M. (2017). Rapid review guidebook. *Natl Collab Cent Method Tools*, 13:25.

HUAI, Q., Hsu, W., Lu, J., Liang, H., Xu, H., and Chen, W. (2021). {XFUSE}: An infrastructure for running filesystem services in user space. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 863–875.

JACKSON, T. W. and SMITH, S. (2012). Retrieving relevant information: traditional file systems versus tagging. *Journal of Enterprise Information Management*, 25(1):79–93.

Jain, S. M. (2022). Hugging face. In *Introduction to transformers for NLP: With the hugging face library and models to solve problems*, pages 51–67. Springer.

KITCHENHAM, B. (2004). Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004):1–26.

LEE, C., Sim, D., Hwang, J., and Cho, S. (2015). {F2FS}: A new file system for flash storage. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pages 273–286.

LEMBKE, J., Roman, P.-L., and Eugster, P. (2022). Defuse: An interface for fast and correct user space file system access. *ACM Transactions on Storage (TOS)*, 18(3):1–29.

LIU, W., Rioul, O., Mcgrenere, J., Mackay, W. E., and Beaudouin-Lafon, M. (2018). Bigfile: Bayesian information gain for fast file retrieval. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–13.

LIU, Y., Li, H., Lu, Y., Chen, Z., and Zhao, M. (2019). An efficient and flexible metadata management layer for local file systems. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*, pages 208–216. IEEE.

OU, J., Shu, J., and Lu, Y. (2016). A high performance file system for non-volatile main memory. In *Proceedings of the Eleventh European Conference on Computer Systems*, pages 1–16.

SINGH, A., Srivatsa, M., and Liu, L. (2007). Efficient and secure search of enterprise file systems. In *IEEE International Conference on Web Services (ICWS 2007)*, pages 18–25. IEEE.

Sinha, S., Kalwani, P., Shah, A., and Gonsalves, J. (2025). High-performance file searching with rust: Parallelized indexing for enhanced computational efficiency. In *2025 IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*, volume 3, pages 1–6. IEEE.

VANGOOR, B. K. R., Agarwal, P., Mathew, M., Ramachandran, A., Sivaraman, S., Tarasov, V., and Zadok, E. (2019). Performance and resource utilization of fuse user-space file systems. *ACM Transactions on Storage (TOS)*, 15(2):1–49.

YANG, F., Kangy, J., Ma, S., and Huai, J. (2018). A highly non-volatile memory scalable and efficient file system. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pages 431–438. IEEE.

YANG, Y., Cao, Q., Yao, J., Jiang, H., and Yang, L. (2020). Batch-file operations to optimize massive files accessing: Analysis, design, and application. *ACM Transactions on Storage (TOS)*, 16(3):1–25.