SPECTRUM OF ENGINEERING SCIENCES
ISSN (E) 3007-3138 (P) 3007-312X VOLUME 4, ISSUE 6, 2026

DEEPSORTENGINE: A LOCAL-FIRST, PRIVACY-PRESERVING INTELLIGENT DESKTOP FILE ORGANIZER USING HYBRID SEMANTIC CLASSIFICATION

Muhammad Basim[*1], Hammad Ahmad Qazi[2], Samiullah[3], Aliha Shahzad[4], Ehram Aylia Awan[5], Abdullah Shahzad[6]

[*1,2,3,5]*Student at HITEC University Taxila*
[4]*Business Assurance Analyst, jazz*
[6]*Senior Software Engineer, Institute CCRIPT Agency*

[*1]basimcr80@gmail.com,[2]qazihammad.x@gmail.com,[3]sim319193@gmail.com,
[4]alihashahzad12@gmail.com, [5]ehramawan35@gmail.com, [6]shahzad.abdullah2012@gmail.com

DOI: https://doi.org/10.5281/zenodo.20624252

Keywords
Desktop file management, local-first software, hybrid classification, ONNX Runtime, semantic search, vector embeddings, privacy-preserving machine learning.

Article History
Received: 07 April 2026
Accepted: 19 May 2026
Published: 10 June 2026

Corresponding Author: *
Muhammad Basim

Abstract
In the modern digital landscape, desktop users accumulate massive quantities of unstructured files, leading to severe digital clutter and diminished productivity. Traditional file managers lack content awareness, requiring laborious manual sorting or brittle, rule-based configurations. To bridge this gap, this paper presents DeepSortEngine, an intelligent, local-first file organization application that automates file sorting through real-time file system monitoring and a hybrid classification pipeline. The proposed system integrates user-learned patterns, deterministic keyword rules, and deep-learning-based vector embeddings to provide adaptive folder recommendations through a non-intrusive accept/reject user workflow. Crucially, to accommodate deployment on consumer-grade hardware with strict resource constraints, the intelligent engine was migrated from an overhead-heavy PyTorch framework to an inference-optimized ONNX Runtime architecture. This optimization yielded a 99.3% reduction in runtime dependency size (from ~2GB to ~15MB) and an 83% decrease in idle memory footprint (from ~300MB to ~50MB), enabling efficient, CPU-only background operations. Furthermore, the architecture introduces a 7-stage hybrid semantic search engine built directly upon an embedded SQLite vector extension (sqlite-vec), enabling context-rich natural language queries under a local-first, privacy-preserving paradigm.

INTRODUCTION

The rapid expansion of the global digital ecosystem has significantly increased the volume of data generated, modified, and stored on personal computing systems. Modern desktop users interact daily with diverse categories of digital assets, including academic documents, financial records, software repositories, multimedia content, and communication files. Despite substantial advancements in computational performance and storage technologies, the fundamental paradigms of desktop file management have remained largely unchanged since their introduction in the 1970s. As a result, users continue to rely on traditional hierarchical folder structures that are increasingly inadequate for managing contemporary data growth [1].

The inefficiency of current file organization practices has become a measurable productivity concern. According to an empirical study by the International Data Corporation (IDC),

knowledge workers spend approximately 2.5 hours per week searching for misplaced or poorly organized files, resulting in nearly 130 hours of lost productivity annually per employee. Beyond productivity loss, excessive digital clutter also contributes to cognitive overload, frustration, and reduced workflow efficiency [2].

The majority of unstructured data accumulation occurs within commonly used directories such as the *Desktop* and *Downloads* folders. Over time, these locations become saturated with heterogeneous files including invoices, lecture notes, downloaded media, temporary scripts, bank statements, and application installers. Since users rarely maintain consistent organizational practices, these directories evolve into highly cluttered repositories that hinder efficient retrieval and management [3].

Recent surveys indicate that 57% of office workers experience anxiety related to disorganized digital files, while 83% report frequent delays when searching for important documents. Additionally, unmanaged *Downloads* folders often contain more than 200 unsorted files, demonstrating the widespread failure of manual organization approaches.

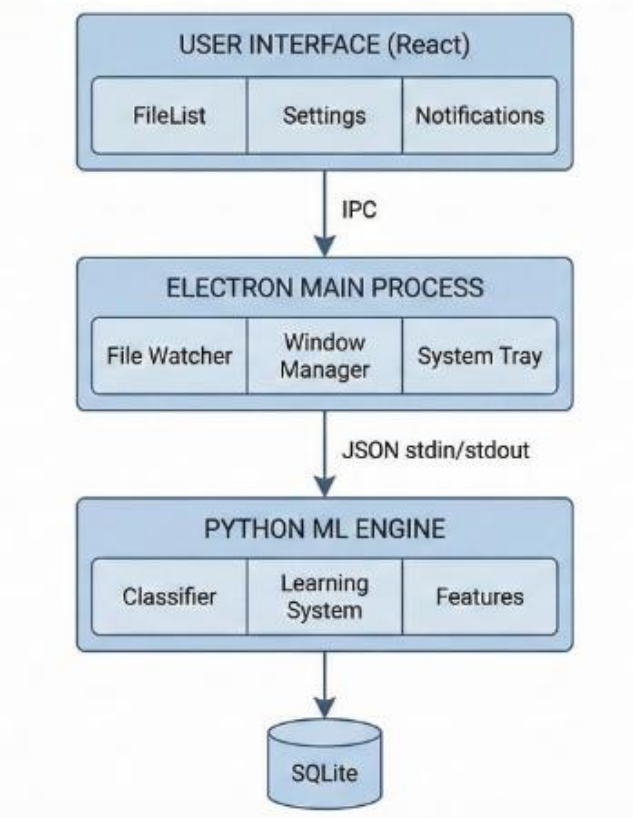

Figure 1 Overview of the proposed system architecture, showing communication between the React-based user interface, Electron main process, Python ML engine, and SQLite database.

Traditional file management systems exhibit several inherent limitations:

1. **High Manual Overhead:**
Users must manually perform repetitive drag-and-drop operations and folder assignments for every new file.

2. **Naming Inconsistency:**
Files are often saved with ambiguous or non-standard naming conventions, reducing searchability and organization accuracy.

3. **Semantic Context Loss:**
Conventional folder hierarchies fail to preserve relationships among related project assets distributed across multiple formats.

4. **Scalability Limitations:**

Manual organizational strategies become increasingly ineffective as the number of files grows over time.

These shortcomings highlight the need for intelligent automation mechanisms capable of reducing user effort while maintaining organizational accuracy [4-5].

Current desktop file organization solutions generally fall into two major categories: cloud-based intelligent systems and deterministic rule-based automation tools. Although both approaches provide partial solutions, neither adequately addresses the combined requirements of privacy, adaptability, usability, and offline operation. Cloud-dependent platforms such as Google Photos and Microsoft OneDrive employ advanced machine learning algorithms to classify and organize files. However, these systems require continuous uploading of user data to remote servers, introducing concerns regarding privacy, internet dependency, subscription costs, and compliance risks associated with sensitive information [6-9].

In contrast, local automation tools such as Hazel and DropIt perform file sorting directly on the client machine through predefined rules and filters. While efficient and privacy-preserving, these systems lack adaptive intelligence and are unable to learn from user behavior. Furthermore, they often require complex rule configuration and regular expression knowledge, making them inaccessible to non-technical users [10]. Consequently, a significant research gap exists for a lightweight, cross-platform, local-first file management system capable of performing intelligent semantic classification and adaptive learning entirely offline. Such a system must combine the privacy advantages of local processing with the intelligent decision-making capabilities of modern machine learning approaches while maintaining low computational overhead suitable for continuous background execution. This research proposes the development of DeepSortEngine, an intelligent on-device file organization framework designed to automate desktop file management through hybrid semantic classification and adaptive learning mechanisms [11-13].

The primary objectives of the proposed system are as follows:

- To implement real-time file system monitoring for immediate detection of newly created or downloaded files.
- To develop a hybrid classification architecture integrating pattern recognition, rule-based filtering, and deep learning techniques.
- To provide a user-in-the-loop verification mechanism that enables accept/reject confirmation and transactional rollback capabilities.
- To ensure strict local-first operation in which all data processing, embedding generation, classification, and storage occur entirely offline.
- To incorporate adaptive learning mechanisms capable of refining classification behavior based on implicit user feedback and correction history.

The effectiveness of the proposed system will be evaluated using measurable engineering benchmarks. These include achieving a 100% real-time file detection rate with sub-second processing latency, obtaining an initial classification accuracy between 70% and 80%, and improving precision to approximately 85%–90% after at least 100 user corrections. Additionally, the system is expected to maintain an idle CPU utilization below 5% and a maximum background memory consumption under 100 MB RAM to ensure lightweight continuous operation [14].

Literature Review

Traditional file management systems, including standard desktop interfaces such as Windows Explorer and Finder, continue to rely heavily on hierarchical directory structures originally conceptualized in the 1970s. These systems require users to manually organize files into folders and subfolders without providing any intelligent understanding of file content or contextual relationships. Their search mechanisms are generally limited to filename matching and metadata filtering, which often becomes inefficient as data volumes increase. Consequently, modern users experience significant difficulty managing large collections of heterogeneous files, particularly within frequently used directories such as Desktop and Downloads folders [15].

Commercial solutions have attempted to address these limitations through automation and machine learning integration. Platforms such as Google Photos demonstrate the effectiveness of deep learning techniques for media categorization, face clustering, and object recognition. By utilizing advanced neural architectures, these systems can automatically organize images and videos with high accuracy. However, such approaches remain fundamentally dependent on cloud infrastructure, requiring users to upload personal data to remote servers. This dependency introduces concerns regarding privacy, internet bandwidth consumption, subscription costs, and regulatory compliance for sensitive information. In addition, their functionality is often restricted to media files rather than broader desktop document ecosystems [16].

Alternatively, local automation utilities such as Hazel and TagSpaces prioritize privacy-preserving offline operation by executing entirely on the client machine. Hazel employs deterministic rule-based mechanisms that monitor directories and trigger actions according to file extensions, naming patterns, or metadata. While efficient and lightweight, these systems require users to manually configure complex rules and cannot adapt dynamically to evolving user behavior. Similarly, TagSpaces provides cross-platform offline organization capabilities through tag-based categorization; however, it relies extensively on manual tagging, which becomes impractical for large-scale or continuously growing datasets. These limitations highlight the inadequacy of purely rule-driven or manually maintained organizational approaches for modern file management requirements [17].

Table I: Comparison of File Classification Techniques

Approach	Techniques	Core Advantages	Primary Disadvantages	DeepSortEngine Integration
Rule-Based Systems	Filename regex, extension checks, keyword matching	Extremely low latency, deterministic execution, absolute interpretability	High brittleness, lack of adaptability, manual overhead	Deployed as the secondary structural pass for deterministic patterns.
Statistical ML	TF-IDF vectors, Naive Bayes, Support Vector Machines (SVM)	Highly performant on small training samples, low overhead	Complete loss of structural syntax and deep word context	Integrated into the fallback pipeline for raw document text classification.
Deep Learning	Transformers, Deep Neural Networks, CNNs	High accuracy, native semantic understanding, handles high ambiguity	High resource requirements, model size bloat, CPU latency	Compact ONNX models deployed for deep semantic similarity routing.

Recent advances in deep learning and semantic representation learning have provided new opportunities for intelligent content-aware file classification. Alec Radford and colleagues introduced Contrastive Language-Image Pre-training (CLIP), a multimodal architecture

capable of mapping textual and visual information into a shared embedding space. This framework demonstrated strong zero-shot classification and semantic retrieval performance by learning relationships between images and natural language descriptions. For textual data, Nils Reimers and Iryna Gurevych proposed Sentence-BERT (SBERT), which extends transformer architectures using siamese network structures to generate dense semantic embeddings. These embeddings enable documents with similar contextual meanings to occupy nearby positions in vector space, thereby supporting efficient semantic search, clustering, and document grouping [18-19].

In parallel, the concept of Local-First Software introduced by Martin Kleppmann and collaborators emphasizes user data ownership, offline availability, and privacy-preserving local computation. Local-first architectures advocate executing application logic and data processing directly on end-user devices rather than relying on centralized cloud services. These principles are particularly relevant for intelligent file management systems, where sensitive personal and organizational documents must remain secure while still benefiting from modern machine learning capabilities.

Existing research on automated file classification demonstrates that different approaches offer distinct trade-offs between performance, interpretability, and semantic understanding. Rule-based systems provide deterministic execution and extremely low latency but suffer from brittleness and lack adaptability when encountering novel patterns. Traditional statistical machine learning techniques, including TF-IDF vectorization, Naive Bayes, and Support Vector Machines (SVMs), perform efficiently on limited datasets with relatively low computational overhead; however, they often fail to capture deeper contextual relationships within documents. In contrast, deep learning architectures such as transformers and convolutional neural networks achieve superior semantic understanding and classification accuracy, particularly for ambiguous or unstructured data, although they typically introduce higher computational and memory requirements[20-23].

Recent studies in adaptive user-interface systems further indicate that incorporating implicit feedback mechanisms significantly improves personalization accuracy. Monitoring user actions such as accept/reject decisions allows systems to gradually align predictions with individual organizational habits without requiring explicit retraining sessions. Motivated by these findings, DeepSortEngine adopts a hybrid architecture that integrates deterministic rule execution, statistical text processing, and deep semantic embeddings within a continuous adaptive feedback framework. This combination aims to provide scalable, privacy-preserving, and context-aware file organization while maintaining lightweight execution suitable for continuous background operation on consumer desktop systems [24-25].

Methodology:

DeepSortEngine was designed as a lightweight, local-first intelligent file organization system capable of operating continuously in the background while maintaining low computational overhead and strong privacy guarantees. The proposed architecture follows a modular multi-layer framework that separates user interaction, system control, machine learning inference, and data persistence into independent operational layers. This separation improves scalability, cross-platform compatibility, maintainability, and fault isolation during runtime execution.

The overall architecture consists of four major layers: the user interface layer, the Electron-based controller layer, the machine learning inference engine, and the local persistence layer. The frontend interface is implemented using React and is responsible for displaying pending file queues, confidence metrics, notification panels, and user configuration settings in real time. The application control layer is implemented using Electron, which manages operating-system-level interactions including background execution, system tray integration, inter-process coordination, and lifecycle management of the machine learning subsystem. The semantic inference engine operates as an isolated Python subprocess responsible for feature extraction, classification, and adaptive learning operations. Persistent storage is managed using SQLite configured in Write-Ahead Logging (WAL) mode to ensure atomic transactions and crash-safe execution.

The operational workflow begins with a real-time file monitoring subsystem implemented through Chokidar. The watcher continuously monitors the Desktop and Downloads directories and captures filesystem events with minimal latency. One of the primary technical challenges in desktop monitoring is avoiding premature processing of files that are still being written to disk during browser downloads or network copy operations. To address this issue, the watcher was configured with a stability verification mechanism using the *awaitWriteFinish* parameter. A file event is propagated to the classification pipeline only after the file size remains unchanged for two consecutive polling intervals, thereby preventing incomplete or temporary files from entering the processing stage. Additionally, an in-memory transactional cache stores recently moved file paths for a short expiration window to eliminate recursive event triggering caused by system-initiated file relocations.

Once a stable file is detected, the feature extraction module collects structural metadata including filename patterns, file extensions, file size, and timestamp information. For textual documents such as PDF and DOCX files, raw textual content is extracted and normalized for semantic analysis. For image-based media files, embedded EXIF metadata is parsed to retrieve contextual information. The extracted features are then forwarded to a hybrid multi-tier classification pipeline designed to combine deterministic logic with semantic machine learning inference.

The first stage of classification utilizes a rule-based engine that evaluates filenames, extensions, and keyword patterns against predefined regular expressions and learned routing templates. This layer provides extremely fast deterministic classification for highly structured files such as invoices, receipts, assignments, and financial documents. However, when the rule engine produces insufficient confidence, the file is passed to the semantic AI classification layer.

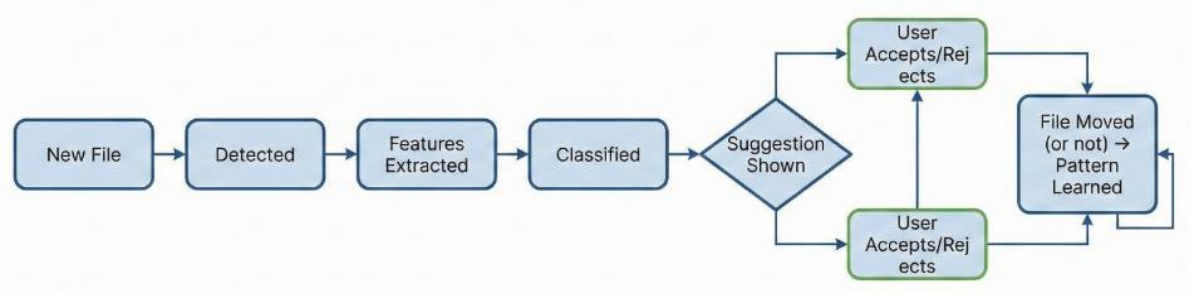

Figure 2 Workflow of the intelligent file organization system, from file detection and feature extraction to classification, user feedback, and adaptive pattern learning.

This layer employs TF-IDF vectorization and transformer-based embedding models to generate semantic representations of document content. Similarity analysis is then performed using cosine similarity against predefined folder embeddings and contextual descriptions to determine the most probable destination category.

The outputs of the learned behavior engine, rule-based classifier, and semantic AI classifier are combined through a weighted decision fusion mechanism. The final confidence score is computed as:

$$F_c = w_1 C_{learned} + w_2 C_{rule} + w_3 C_{ai}$$

where the operational weights satisfy the normalization condition:

$$\sum_{i=1}^{3} w_i = 1.0$$

The experimentally selected values are configured as $w_1 = 0.4$, $w_2 = 0.3$, and $w_3 = 0.3$, giving greater influence to learned user behavior while preserving deterministic structural matching and semantic inference balance. The resulting confidence value is presented within the user interface to allow transparent user verification before execution.

To support personalization, DeepSortEngine incorporates a continuous adaptive learning mechanism. User interactions are treated as

implicit feedback signals that dynamically modify routing preferences stored within the local database. When a user accepts a suggested destination, the associated routing pattern is strengthened through incremental confidence adjustment. Conversely, rejected suggestions reduce the routing confidence score, gradually disabling inaccurate patterns over time. This adaptive mechanism enables the system to align organizational behavior with individual user habits without requiring explicit model retraining sessions.

Persistent data management is implemented using two normalized relational tables within the SQLite database. The *Patterns* table stores dynamically learned routing patterns, destination paths, confidence coefficients, activation frequencies, and update timestamps. The *Events* table functions as an audit ledger containing execution timestamps, source file paths, transaction actions, destination directories, and user validation states. This structure provides traceability, rollback support, and analytical monitoring while maintaining lightweight local storage requirements.

To establish secure and efficient communication between the asynchronous Electron runtime and the synchronous Python inference engine, a bidirectional JSON-RPC communication layer was implemented over standard input/output pipes. Each request payload contains a unique UUID identifier, execution method token, and serialized parameter object. The Electron controller maintains active promise mappings for request tracking and response synchronization. To ensure system stability, all inference operations are protected by a strict timeout mechanism. If a machine learning task exceeds the permitted execution interval due to corrupted or unsupported files, the subprocess is automatically terminated and restarted without affecting the primary application runtime.

The operational lifecycle of a file within DeepSortEngine follows a strictly contained local processing pipeline:

File Detection → Feature Extraction
→ Classification Pipeline
→ User Validation
→ State Synchronization

Throughout this workflow, strict privacy boundaries are maintained. All data processing, storage, embedding generation, and machine learning inference occur exclusively on the local client machine. Files are never uploaded to external servers, and the Python subprocess executes without network access privileges, ensuring complete prevention of external data transmission. Furthermore, file relocation operations are performed using atomic local filesystem transactions to guarantee consistency and eliminate unnecessary file duplication.

Core Innovation:

One of the primary innovations of DeepSortEngine lies in its transition from heavyweight deep learning frameworks to an optimized ONNX-based inference architecture. Early development versions relied on transformer implementations using PyTorch and HuggingFace libraries for semantic embedding generation. Although these frameworks provided accurate semantic classification capabilities, they introduced severe limitations that made them unsuitable for a continuously running desktop utility. The original implementation suffered from excessive dependency sizes exceeding multiple gigabytes, prolonged startup times caused by model initialization overhead, and high idle memory consumption that significantly affected system responsiveness.

To overcome these constraints, the inference subsystem was migrated to ONNX Runtime. This migration enabled transformer models to be converted into highly optimized graph representations suitable for lightweight CPU execution. As a result, the runtime dependency footprint was drastically reduced while application startup latency and memory utilization were significantly improved. Unlike traditional deep learning environments that prioritize training flexibility, the ONNX Runtime framework focuses specifically on efficient inference execution, making it highly suitable for real-time desktop applications operating under constrained system resources.

A centralized model management architecture was introduced to further optimize runtime behavior. Instead of loading all machine learning assets during application startup, DeepSortEngine employs lazy loading mechanisms that initialize models only when semantic inference is explicitly required. This approach allows the graphical interface and

monitoring services to launch instantly without waiting for transformer initialization. Additionally, all inference sessions are managed through controlled acquisition and release procedures that safely handle resource allocation, exception recovery, and memory cleanup. Thread-safe reference tracking mechanisms ensure that models remain resident only while actively in use, after which they become eligible for automatic eviction to preserve low background memory consumption. The inference environment is further optimized for consumer-grade CPUs through controlled thread allocation and graph-level optimization strategies, eliminating the requirement for dedicated GPU acceleration or external driver dependencies.

Another major innovation of DeepSortEngine is the integration of a local Hybrid Semantic Search Engine capable of performing natural language retrieval across indexed files. Traditional desktop search systems rely heavily on exact filename or keyword matching, which fails when users cannot remember precise file names or directory locations. DeepSortEngine instead allows users to search using contextual phrases such as "tax forms from last year" or "lecture slides about neural networks." The search engine combines semantic vector similarity with deterministic keyword boosting to improve both contextual understanding and precision.

The search pipeline begins by converting user queries into normalized semantic embeddings using transformer-based ONNX models. These embeddings are compared against locally stored document vectors using cosine similarity retrieval within the embedded SQLite vector extension. Raw similarity measurements are normalized into bounded semantic relevance scores:

$$\text{Similarity} = 1.0 - \left(\frac{\text{Distance}^2}{2.0}\right)$$

To maintain precision for short or exact searches, semantic similarity is supplemented with a keyword boosting layer that prioritizes direct filename matches, substring occurrences, and token-level overlaps within document content. The final retrieval score is computed through a dynamic weighting mechanism that adaptively balances semantic understanding and exact keyword matching:

$$\text{Final Score} = \alpha \cdot \text{Semantic Score} + \beta \cdot \text{Keyword Score}$$

For short queries containing only a few words, the system increases the contribution of keyword matching to prioritize rapid deterministic retrieval. For longer contextual queries, the weighting shifts toward semantic similarity to improve understanding of conceptual meaning and intent. This adaptive balancing strategy enables DeepSortEngine to support both precise filename lookups and context-aware natural language searches within a unified retrieval framework.

Unlike conventional semantic search systems that depend on external vector databases or cloud indexing infrastructure, DeepSortEngine stores vector embeddings directly inside the local SQLite instance using the sqlite-vec extension. This architecture eliminates the need for additional server processes while preserving atomic transactional consistency between relational metadata and semantic vectors. The use of Write-Ahead Logging further enables safe concurrent read and write operations during continuous background indexing. To improve responsiveness during rapid user input, the search engine also incorporates an in-memory Least Recently Used (LRU) caching mechanism that stores recently executed search structures and invalidates stale entries whenever the database index changes.

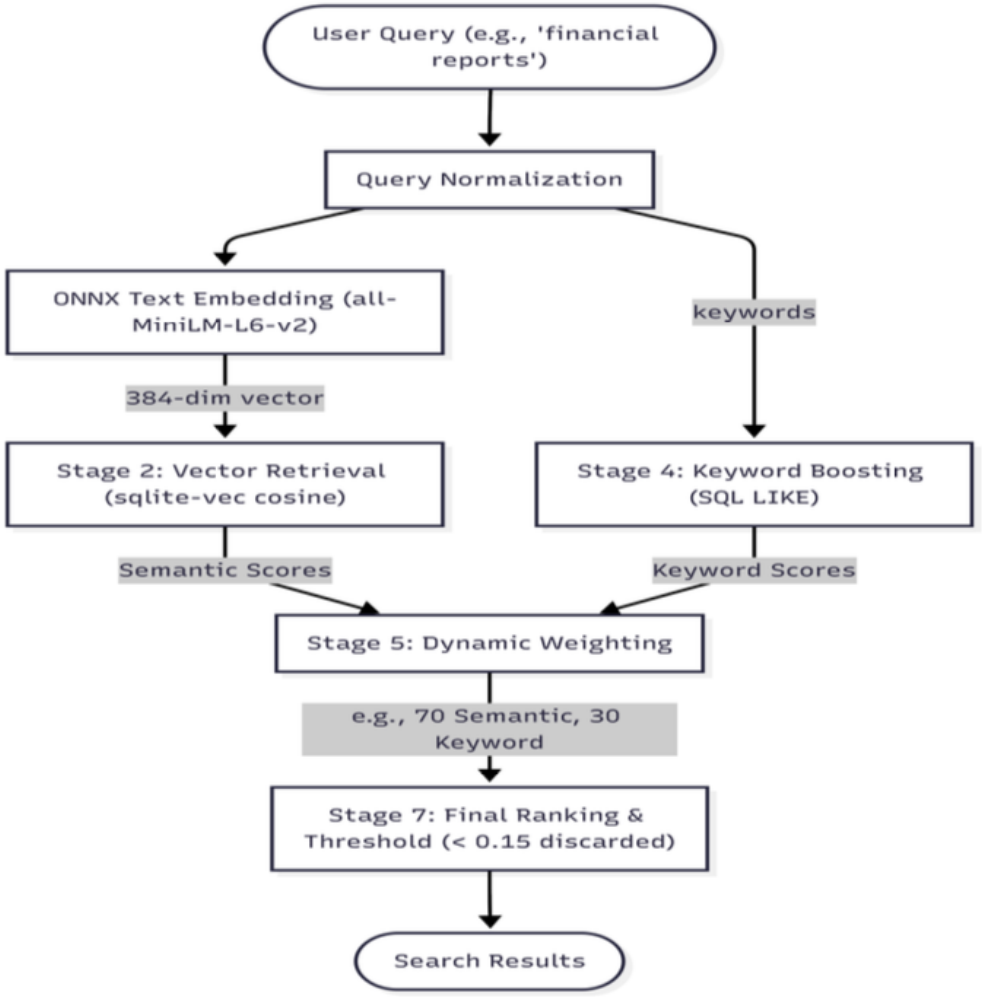

Figure 3 Hybrid semantic–keyword retrieval and ranking pipeline used for document search.

Another significant architectural contribution is the introduction of content-hash-based indexing to establish stable file identity independent of mutable filesystem paths. Traditional file managers identify files primarily through directory paths and filenames, making them vulnerable to tracking failures whenever files are renamed or relocated. DeepSortEngine instead generates unique SHA-256 content hashes from the raw binary representation of each indexed file. This content-addressable model enables reliable duplicate detection, resilient tracking across directory moves, and automatic re-indexing whenever a file's internal content changes.

The use of content hashing also strengthens the adaptive learning subsystem by ensuring that learned behavioral patterns remain associated with the actual file content rather than unstable path strings. Additional hardening mechanisms further improve learning stability by filtering volatile tokens, isolating category boundaries, and enforcing strict confidence thresholds before presenting automated routing suggestions. Specifically, DeepSortEngine suppresses low-confidence recommendations to reduce incorrect classifications and maintain user trust in the automated organization process. The minimum recommendation threshold is enforced as:

$$C_{min} \geq 0.55$$

Through the combination of lightweight ONNX inference, adaptive semantic retrieval, local vector persistence, and content-addressable indexing, DeepSortEngine introduces a novel desktop file organization framework that balances intelligent automation, privacy preservation, and resource-efficient execution within a fully offline environment.

Results and Discussions:

A. Performance Benchmarks

To evaluate the optimization gains achieved by migrating from the PyTorch inference framework to the ONNX Runtime engine, empirical benchmarks were collected on standard consumer equipment (Intel Core i5, 8GB RAM). The comparative findings are detailed in Table II:

Table II: Runtime Optimization Metrics

Performance Metric Target	PyTorch Architecture (Baseline)	ONNX Runtime Architecture	Net Architectural Improvement
Runtime Dependency Footprint	$\sim 2\text{ GB}$	$\sim 15\text{ MB}$	**99.3% Storage Reduction**
Cold-Start Model Compilation	$5.0 - 10.0\text{ seconds}$	$< 1.0\text{ second}$	**~10x Startup Acceleration**
Inference Latency (per query)	$\sim 50\text{ ms}$	$\sim 5\text{ ms}$	**~10x Computation Speedup**
Idle Memory Consumption	$\sim 300\text{ MB}$	$\sim 50\text{ MB}$	**83.3% RAM Footprint Savings**
Hardware Compute Prerequisite	GPU Acceleration Recommended	CPU-Only Operation Optimized	**Eliminated External Driver Dependencies**

The results show that the ONNX Runtime framework successfully addresses the resource overhead constraints of the early builds. The idle memory footprint was brought down to ~50MB, well under the 100MB success criteria and allowing the tool to run efficiently in the background without affecting core system performance.

B. Search Retrievability and UI Responsiveness

The integration of the custom in-memory LRU cache with the sqlite-vec extension minimized search delays during rapid multi-character entries. While direct vector math queries across un-cached rows take around 5 milliseconds, cache hits return results in under 1 millisecond, providing a responsive interface for user interaction.

C. Architectural Robustness and Regression Testing

To guarantee long-term system stability across varying edge environments, a continuous 18-test automated regression suite was developed. This suite tests four critical aspects of the local database and pipeline execution layers:

1. *Legacy Pattern Migration:* Verifies the structural integrity of user data models during automated transitions from v1 to v3 schemas without data loss.
2. *Confidence Threshold Enforcement:* Confirms that the hybrid classification engine blocks routing suggestions whose score falls below the required 0.55 gate.
3. *Base Path Nesting Prevention:* Validates structural boundaries to prevent the file monitor from generating circular paths or moving parent directories into their own sub-folders.
4. *Nested Path Purge Migration:* Confirms that deleting a parent directory safely cleans up

child metadata references across both the relational tables and the vector index spaces.

VII. Conclusion and Future Work

This paper presented DeepSortEngine, an intelligent, privacy-preserving desktop file organization application that operates entirely on local device edge partitions. By combining user pattern induction, deterministic keyword regular expressions, and deep semantic transformer embeddings, the architecture delivers adaptive folder recommendations through a non-intrusive accept/reject verification workflow. Migrating the core inference pipeline to the ONNX Runtime framework successfully resolved the resource constraints typical of deep learning setups, reducing background memory consumption to ~50MB and installer requirements to ~15MB. Furthermore, integrating vector search directly into SQLite via sqlite-vec enables context-rich, natural language querying while ensuring absolute user data privacy.

Future development paths will focus on expanding DeepSortEngine's real-time monitoring and sorting capabilities beyond the default ingress folders to cover the user's entire file system. Additionally, future iterations will explore the integration of quantized local vision models to enable unified, content-aware classification of complex local media structures entirely offline.

REFERENCES:

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., ... &Sutskever, I. (2021). Learning transferable visual models from natural language supervision. In International Conference on Machine Learning (pp. 8748-8763). PMLR.

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing.

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In NAACL-HLT.

Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255).

Kleppmann, M., Wiggins, A., Van Hardenberg, P., & McGranaghan, M. (2019). Local-first software: You own your data, in spite of the cloud. In Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software.

Ahmed, D., Dillshad, V., Danish, A. S., Jahangir, F., Kashif, H., & Shahbaz, T. (n.d.). *Enhancing Home Automation through Brain-Computer Interface Technology*. Retrieved http://xisdxjxsu.asia

Bint-E-Asim, H., Iqbal, S., Danish, A. S., Shahzad, A., Huzaifa, M., & Khan, Z. (n.d.). *Exploring Interactive STEM in Online Education through Robotic Kits for Playful Learning* (Vol. 19). Retrieved http://xisdxjxsu.asia

Danish, A. S., Khan, Z., Jahangir, F., Malik, A., Tariq, W., Muhammad, A., & Khan, Y. (n.d.). *Exploring the Effectiveness of Augmented Reality based E-Learning Application on Learning Outcomes in Pakistan: A Study Utilizing VARK Analysis and Hybrid Pedagogy*. Retrieved http://xisdxjxsu.asia

Danish, A. S., Malik, A., Lashari, T. A., Javed, M. A., Asim, H. B., Muhammad, A., & Khan, Y. (n.d.). *Evaluating the User Experience of an Augmented Reality E-Learning Application for the Chapter on Work and Energy using the System Usability Scale*. Retrieved http://xisdxjxsu.asia

Danish, A. S., Waheed, Z., Sajid, U., Warah, U., Muhammad, A., Khan, Y., & Akram, H. (n.d.). *Exploring Temporal Complexities: Time Constraints in Augmented Reality-Based Hybrid Pedagogies for Physics Energy Topic in Secondary Schools*. Retrieved http://xisdxjxsu.asia

Faizan Hassan, M., Mehmood, U., Samad Danish, A., Khan, Z., Muhammad Yar Khan, A., & Muneeb Asad, R. (n.d.-a). *Harnessing Augmented Reality for Enhanced Computer Hardware Visualization for Learning*. Retrieved http://xisdxjxsu.asia

Faizan Hassan, M., Mehmood, U., Samad Danish, A., Khan, Z., Muhammad Yar Khan, A., & Muneeb Asad, R. (n.d.-b). *Harnessing Augmented Reality for Enhanced Computer Hardware Visualization for Learning*. Retrieved http://xisdxjxsu.asia

Khan, J., Jalil, Z., Ali, S., & Samad Danish, A. (2019). Implementation of Smart Aquarium System Supporting Remote Monitoring and Controlling of Functions using Internet of Things. In *Journal of Multidisciplinary Approaches in Science* (Vol. 9). JMAS.

Lashari, T. A., Danish, A. S., Lashari, S. A., Sajid, U., Khan, Z., & Saare, M. A. (n.d.). *Impact of custom built videogame simulators on learning in Pakistan using Universal Design for Learning*. Retrieved http://xisdxjxsu.asia

Muhammad, A., Khan, Y., Danish, A. S., Aizaz, F., Bilal, H., Shahrose, A., Asad, R. M., & Rafiq, M. T. (n.d.). *Navigating Cybersecurity in Global Software Development: Insights, Challenges, and Practices*. Retrieved http://xisdxjxsu.asia

Muhammad, A., Khan, Y., Danish, A. S., Haider, I., Batool, S., Javed, M. A., & Tariq, W. (n.d.). *Enhancing Social Media Text Analysis: Investigating Advanced Preprocessing, Model Performance, and Multilingual Contexts*. Retrieved http://xisdxjxsu.asia

Muhammad Yar Khan, A., Shaheen, S., Samad Danish, A., Warah, U., -UR-Rehman, O., & Faizan Hassan, M. (n.d.-a). *CISCO Packet Tracer Enterprise Level Architecture using the Concept of SDN*. Retrieved http://xisdxjxsu.asia

Muhammad Yar Khan, A., Shaheen, S., Samad Danish, A., Warah, U., -UR-Rehman, O., & Faizan Hassan, M. (n.d.-b). *CISCO Packet Tracer Enterprise Level Architecture using the Concept of SDN*. Retrieved http://xisdxjxsu.asia

Samad Danish, A., Attique Khan, M., Ijaz, S., & -UR-Rehman, O. (n.d.). *Exploring Student Morale through Technology Acceptance Model in Higher Education: A Study of AR-Based E-Learning Application*. Retrieved http://xisdxjxsu.asia

Samad Danish, A., Noor, N., Hamid, Y., Ali Khan, H., Muneeb Asad, R., & Muhammad Yar Khan, A. (n.d.). *Augmented Narratives: Unveiling the Efficacy of Storytelling in Augmented Reality Environments*. Retrieved http://xisdxjxsu.asia

Samad Danish, A., Warah, U., Sajid, U., Adnan Javed, M., & Muhammad Yar Khan, A. (n.d.). *Evaluating the Feasibility and Resource Implications of an Augmented Reality-Based E-Learning Application: A Comprehensive Research Analysis*. Retrieved http://xisdxjxsu.asia

Spectrum of Engineering Sciences ISSN (e) 3007-3138 (p) 3007-312X. (n.d.). https://doi.org/10.5281/zenodo.17365636

Talha Rafiq, M., Osama Habib Gilani, S., Hina Habib Gilani, S., Samad Danish, A., & Muhammad Yar Khan, A. (n.d.). *Augmented Reality Interface for Seamless Control and Management of IoT Devices in Unity Engine*. Retrieved http://xisdxjxsu.asia

Tariq, W., Ali, I., Naeem, H., Batool, S., Faizan Hassan, M., Samad Danish, A., & Muhammad Yar Khan, A. (n.d.). *Enhancing Educational Outcomes through Augmented Reality: A Case Study on Newton's Laws of Motion*. Retrieved http://xisdxjxsu.asia

Yar Khan, A., Samad Danish, A., Hamid, Y., Khan, F., & Ali Kiani, S. (n.d.). *Unraveling Pakistan's Network Landscape-Legacy Structures vs. SDN Paradigms in the Internet Age in IoT Architecture*. Retrieved http://xisdxjxsu.asia