Spacedrive: Architecture of a Content-Aware Virtual File System

A Local-First VDFS for Unifying Data Across Distributed Devices

James Mathew Pine
james@spacedrive.com
Spacedrive Technology Inc.
Vancouver, British Columbia, Canada

Abstract

Data fragmentation across devices and clouds hinders cohesive file management. Spacedrive implements a local-first [?] Virtual Distributed File System (VDFS) that unifies data views while preserving original file locations. Unlike cloud-centric alternatives, it supports offline operation, minimizes reliance on centralized services, and is applicable to both single-user and organizational deployments.

Core features include a unified data index for low-latency search, content-aware deduplication, and transactional cross-device operations. This index also supports a local AI component for natural language queries and assistive automation.

This paper describes the Spacedrive v2 architecture, including content-aware addressing, transactional previews, and a hybrid synchronization model that avoids distributed consensus by combining simple state-replication for device-authoritative data with a lightweight, HLC-ordered log for shared metadata. The Content Identity system enables deduplication and redundancy tracking, while AI integration provides semantic search and optional automated actions. We also present a cloud deployment in which backends participate as standard P2P devices, reducing client–server asymmetry.

CCS Concepts

• **Information systems → Hierarchical storage management**; **Query representation**; • **Software and its engineering → Software architectures**.

Keywords

Virtual Distributed File System, VDFS, AI-Native Architecture, Natural Language File Management, Semantic Search, Data Synchronization, Tiered Storage, Local-First AI, Rust

ACM Reference Format:
James Mathew Pine. 2025. Spacedrive: Architecture of a Content-Aware Virtual File System: A Local-First VDFS for Unifying Data Across Distributed Devices. In *Proceedings of Spacedrive Whitepaper (Spacedrive '25).* ACM, New York, NY, USA, 35 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

The proliferation of computing devices and cloud services has fragmented our digital lives, scattering assets across incompatible ecosystems [? ?]. Knowledge workers waste up to 25% of their time searching for files, with creative professionals particularly affected [?]. Existing solutions force a choice between cloud convenience and local control, with none offering a unified, content-aware approach.

Spacedrive addresses this by implementing a **local-first Virtual Distributed File System (VDFS)** that unifies data management while files remain in their original locations [?]. Unlike cloud-centric services, it supports offline operation, preserves local control, and applies to both single-user and multi-user deployments.

The architecture is built on four foundational principles that solve traditionally hard problems in distributed systems:

- **Unified Data Model: The Content-Aware VDFS** (Section 4.1): A single, virtual layer that unifies all data across devices by focusing on the content of files, not just their names and locations. Built on an Entry-Centric Model (Section 4.1.2) where every file and folder is a first-class object capable of holding rich metadata instantly, and a Content Identity System (Section 4.2) that uses adaptive hashing to identify and track unique file content, enabling deduplication, redundancy tracking, and resilient content-based addressing.
- **Safe & Predictable Operations: The Transactional Action System** (Section 4.4): A system designed to make file operations safe and transparent. Every change to the VDFS is treated as a transaction that is simulated and approved before execution, leveraging the complete VDFS index to run in-memory simulations of proposed actions with previews of outcomes, including conflicts or storage issues.
- **Resilient Synchronization: A Leaderless Hybrid Model** (Section 4.5.1): A synchronization model that eliminates leader-based bottlenecks by separating data into distinct domains. The filesystem index, being device-authoritative, is replicated across all devices, enabling File Sync to operate through efficient index queries rather than filesystem scanning. Concurrently modified user metadata is merged deterministically using a lightweight, per-device change log ordered by a Hybrid Logical Clock (HLC).
- **AI Agent Architecture** (Section 4.6): An architecture where specialized AI agents are implemented as WASM extensions, each with domain-specific knowledge and persistent memory systems. These agents observe the VDFS index, react to events, and propose actions using the same transactional model as human-initiated operations.

We report a reference implementation in Rust. On consumer hardware, we observe sub-100 ms semantic search latency, synchronization via the Iroh networking stack, and an approximate memory footprint of 150 MB for libraries exceeding one million files.

This paper details Spacedrive's architecture as a production system implemented in Rust. We show how domain separation and content awareness enable cross-device deduplication, semantic search, storage tiering, and synchronization without distributed consensus.

Spacedrive adapts to mobile device constraints through built-in resource management. On battery power, it reduces CPU usage. It respects data limits and platform background restrictions. The system runs efficiently on both desktops and smartphones, maintaining performance across all devices.

Spacedrive applies local-first architecture [?] principles at scale. Traditional enterprise systems sacrifice user experience for control. Consumer tools lack business security features. Spacedrive aims to provide both. It scales from personal use to enterprise deployment while maintaining security, data sovereignty, and compliance.

We also outline a cloud service built on this architecture. The cloud backend runs as a standard Spacedrive device. Users pair with it using the same peer-to-peer protocols as local machines. This hybrid approach offers cloud-hosted availability while retaining local-first privacy and control properties.

2 Related Work

Spacedrive builds upon decades of research in distributed file systems, personal information management, and content-addressable storage [?]. We position our work within this landscape to highlight our unique contributions.

2.1 Traditional Cloud Sync Services

Commercial cloud storage services (Dropbox [?], Google Drive, iCloud) provide basic file synchronization but suffer from platform lock-in and lack content-addressing. These services treat files as opaque blobs, missing opportunities for deduplication and semantic understanding. Unlike Spacedrive, they require continuous internet connectivity and centralize user data on corporate servers.

2.2 Distributed File Systems

Research systems like IPFS [?] and production systems like Ceph [?] demonstrate the power of content-addressable storage. However, their complexity and resource requirements make them unsuitable for personal use. IPFS requires understanding of cryptographic hashes and peer-to-peer networking, while Ceph targets datacenter deployments. Spacedrive adopts content-addressing principles while hiding complexity behind familiar file management interfaces, drawing inspiration from systems like LBFS [?] that pioneered content-defined chunking for efficient network transfers.

2.3 Virtual Distributed File Systems in the Datacenter

The concept of a Virtual Distributed File System (VDFS) has been explored in the context of large-scale data analytics. Alluxio (formerly Tachyon) [?] introduced a memory-centric VDFS designed to sit between computation frameworks like Apache Spark and various storage systems (e.g., HDFS, S3). Alluxio's primary goal is to accelerate data analytics jobs by providing a consolidated, high-throughput data access layer, effectively decoupling computation from storage in a datacenter environment.

While Spacedrive shares the VDFS terminology, its architectural goals and target domain are fundamentally different. Where Alluxio optimizes for performance in large, multi-tenant analytics clusters, Spacedrive is designed as a local-first, privacy-preserving data space for an individual's complete digital life. Spacedrive's innovations in universal addressing (`SdPath`), an entry-centric model with immediate metadata, and Library Sync are tailored to the challenges of personal data fragmentation across a heterogeneous collection of consumer devices, a problem space distinct from the performance and data-sharing challenges in large-scale analytics that Alluxio addresses.

2.4 Personal Knowledge Management

Tools like Obsidian and Logseq excel at managing structured knowledge through markdown files but lack general file management capabilities. They demonstrate the value of local-first architectures [?] and portable data formats, principles that Spacedrive extends to all file types. Our work generalizes their approach from text-centric knowledge graphs to general-purpose file management.

2.5 Self-Hosted Solutions

Projects like Nextcloud provide self-hosted alternatives to commercial cloud services but retain client-server architectures that complicate deployment and maintenance. They require dedicated servers and technical expertise, limiting adoption. Spacedrive's peer-to-peer architecture eliminates server requirements while providing similar capabilities through direct device communication.

2.6 Semantic File Systems

Academic projects exploring semantic file organization date back to the Semantic File System [?] and Presto [?]. While these demonstrated the value of content-based organization, they predated modern AI capabilities. Spacedrive applies contemporary machine learning, enabling natural language queries and automated tasks that were not feasible in earlier systems.

2.7 Comparative Analysis

Table 1 summarizes how Spacedrive advances beyond existing systems by combining their strengths while addressing their limitations.

Our work synthesizes insights from these domains while addressing their individual limitations, creating an integrated system that is effective, private, and accessible to non-technical users.

2.8 Command-Line Data Movers

Command-line utilities like rclone [?] excel at performing scriptable data transfers between a wide variety of storage backends. These tools are effective for one-off data moving tasks and are a staple for technical users. However, their fundamentally stateless architecture presents limitations that Spacedrive's stateful, persistent model addresses.

System	Architecture	Target Users	Key Innovation	Primary Limitation	Privacy Model
Dropbox/ iCloud	Client-Server	Consumers	Simple sync	No content addressing, vendor lock-in	Cloud-centralized
IPFS	P2P DHT	Developers	Content addressing	Complex for consumers, no AI	Public by default
Ceph	Distributed cluster	Enterprises	Scalable storage	Datacenter-focused, high overhead	Configurable
Alluxio	Memory-centric VDFS	Analytics teams	Unified data access	Not for personal files	Enterprise-managed
Nextcloud	Self-hosted server	Tech-savvy users	Data sovereignty	Requires dedicated server	Self-hosted private
Spacedrive	**Local-first P2P**	**Everyone**	**AI-native VDFS**	**Higher resource usage than simple browsers**	**Local-first E2E**

Table 1: Comparison of Spacedrive with existing systems, expanded with privacy models for completeness.

Each time a command is executed, a stateless tool must re-query both the source and destination to determine the necessary changes. Spacedrive, in contrast, operates as a data orchestrator rather than just a data mover. It maintains a continuously updated VDFS index, enabling it to know the state of files across locations in real time. Spacedrive leverages its persistent state to perform synchronization through global content-aware deduplication, optimal path routing for transfers, and a "preview-then-commit" transactional model that enhances safety and reliability. While rclone is a widely used tool for explicit data transfer, Spacedrive operates at a higher level of abstraction, integrating synchronization as a native, persistent feature of a cohesive data space. The system's approach to maintaining index integrity during offline periods is detailed in Section 4.3.4.

Extensibility Models. Unlike systems that require native plugins (Finder, Nautilus) or rely on scripting languages (Obsidian, VS Code), Spacedrive employs a consistent WebAssembly-based extensibility model. All extensions—from simple content type handlers to complex cloud storage integrations—run in a secure WASM sandbox. This provides both power and safety without the complexity of managing multiple extension systems.

Cloud Storage Approaches. Unlike traditional sync clients that duplicate data locally, Spacedrive treats cloud storage as just another volume through direct indexing (detailed in Section 4.10, including traditional protocols like FTP and SMB).

3 Learning from the Past: Architectural Evolution from Spacedrive v1

Spacedrive v2 represents a complete architectural reimplementation designed to fulfill the original vision on a more reliable and scalable foundation. The initial version, first open-sourced in 2022, validated the core premise of an integrated VDFS for personal data with significant community interest. However, as development progressed through early 2025, several foundational architectural challenges emerged that ultimately necessitated this rewrite, drawing lessons from the evolution of distributed systems and CRDTs [**?**].

3.1 Key Challenges in the Original Architecture

The development of v1 provided invaluable insights that shaped the v2 architecture. Building Spacedrive required pioneering work in an immature ecosystem—Rust application development was uncommon at the time, Tauri was in its infancy, and many essential libraries simply didn't exist. Our engineers contributed extensively upstream to Tauri, Iroh, and even the QUIC protocol, with Spacedrive serving as a real-world model for building complex Rust applications. We developed critical infrastructure including prisma-client-rust and rspc, which became foundational tools for the broader Rust ecosystem. However, this pioneering approach created architectural challenges: separate systems for indexed and non-indexed files that couldn't interoperate, tight frontend/backend coupling through query invalidation patterns, and the complexity overhead of custom CRDT-based synchronization. The experience highlighted the importance of unified addressing systems, streamlined job architectures, and the risks of maintaining custom infrastructure when mature alternatives emerge.

3.2 Spacedrive v2 as an Architectural Solution

The v2 architecture directly addresses these challenges. The universal **SdPath** addressing system eliminates the dual file system problem entirely. A decoupled **Event Bus** provides reliable state propagation, and a simpler domain-separated **Library Sync** model ensures consistency without CRDT complexity by employing a leaderless, hybrid architecture. This model uses efficient state-based replication for device-authoritative data and a lightweight, HLC-ordered log for shared metadata, eliminating single points of failure. The new job system reduces boilerplate by over 90%, while a consolidated networking layer powered by **Iroh** unifies all P2P communication, improving reliability and connection performance [**?**]. The technology stack was also modernized, replacing abandoned libraries with community-trusted alternatives like **SeaORM**.

By learning from real-world challenges of the initial version, Spacedrive v2 delivers on the original promise with an architecture that is simpler, more fault-tolerant, and designed for the long term.

Extensibility Lessons. Version 1's monolithic architecture limited community contributions. Version 2's single WASM plugin model enables an active ecosystem while maintaining security and stability. All extensions—from content type handlers to cloud storage providers—run in the same secure sandbox, simplifying development and distribution.

4 The Spacedrive Architecture

Key Takeaways

- **Core Innovation**: Integrated virtual layer that makes distributed storage feel like a single filesystem through content-aware addressing

- **Key Components**: VDFS model with SdPath addressing • Entry-centric metadata • Content Identity deduplication • Transactional Actions • AI-native design
- **Design Philosophy**: Local-first for privacy, peer-to-peer for resilience, AI-assisted analysis—while preserving user control

Spacedrive's architecture represents a new approach to how personal data is managed across devices. Rather than treating files as isolated entities scattered across different storage systems, Spacedrive creates a cohesive virtual layer that provides consistent access, metadata and analysis, and cross-device synchronization. This section presents the core architectural components that enable this vision.

4.1 The VDFS Model

At the core of Spacedrive is a set of abstractions that model a user's data not as a collection of disparate file paths, but as a cohesive, integrated Library with content-aware relationships.

4.1.1 The Library: A Portable Data Container. Rather than managing scattered databases and configurations, Spacedrive organizes everything into self-contained **Libraries**. Each Library is a `.sdlibrary` directory that functions as a complete, portable data container:

A Spacedrive Library is organized as a self-contained directory with four key components:

- **Configuration File**: Library settings and device registry
- **Metadata Database**: Complete file index with relationships and tags
- **Thumbnail Cache**: Organized storage for instant previews
- **Concurrency Protection**: Safe multi-device access control

This portable structure means that backing up your entire organizational system—all metadata, tags, and file relationships—is as simple as copying a single directory. The Library acts as a complete catalog of your digital life, preserving your organizational work even if the actual files are distributed across multiple devices and locations.

This design provides several critical advantages: its self-contained nature simplifies backup and migration, and sharing involves sending the complete Library. From a security perspective, this architecture ensures that each device in a paired group builds and controls its own library database, enabling granular scoping of locations and indexed data to local or select devices, thereby enhancing data sovereignty and limiting exposure.

4.1.2 The Entry-Centric Data Model. The fundamental unit within a Library is the **Entry**—a universal representation that treats files and directories as first-class, queryable objects. This design, building upon early work in document-centric computing [**?**], enables the VDFS to maintain rich metadata and relationships independently of the underlying filesystem.

Uniform Representation: Unlike traditional filesystems where directories and files have fundamentally different representations, every filesystem entity becomes an Entry with consistent properties:

Field	Purpose
Unique ID	Globally unique, immutable identifier
Universal Path	Complete location addressing (SdPath)
Parent ID	Direct hierarchical relationship
Metadata ID	Links to user-assigned metadata
Content ID	Links to content identity (for deduplication)
Discovery Time	Temporal tracking

Table 2: Entry data model

Architectural Benefits: This entry-centric approach provides several key advantages:

- **Metadata Independence**: User metadata (tags, notes) persists regardless of filesystem changes. Moving or renaming files preserves all organizational work.
- **Universal Queryability**: Every entity can be searched, filtered, and organized through the same query interface, whether it's a file, directory, or even ephemeral content.
- **Relationship Tracking**: The parent ID creates an explicit graph structure that enables O(1) hierarchical queries through the closure table (Section 4.3), replacing slow path-based traversals.
- **Content Separation**: The optional content ID links entries to the deduplication system without coupling file identity to content analysis. An entry can exist and be organized before its content is analyzed.

Ephemeral Mode: The indexing engine extends this model to support temporary, in-memory Entry records for unindexed content. When browsing external drives or remote devices, ephemeral entries are generated on-the-fly and presented alongside persisted entries, creating a unified view. Users can tag and organize these files immediately; if an entry lacks a parent Location, it persists in the Library with its metadata intact, demonstrating how the entry-centric model handles both permanent and transient data uniformly.

4.1.3 Semantic Tagging. Spacedrive employs a graph-based tagging system that supports semantic organization while maintaining ease of use. This system recognizes that human organization relies on context, relationships, and multiple perspectives—capabilities that traditional flat tagging systems cannot provide.

Contextual Tag Design

The tagging system provides flexibility:

- **Polymorphic Naming**: Tags embrace natural ambiguity, allowing multiple "Project" tags differentiated by their semantic context rather than forced uniqueness
- **Unicode-Native**: Full international character support enables native-language organization without ASCII constraints
- **Semantic Variants**: Each tag maintains multiple access points—formal names, abbreviations, and contextual aliases

Graph-Based Organization Model

The system implements a directed acyclic graph (DAG) structure using optimized database patterns for millisecond-scale hierarchy traversal:

Context Resolution: When multiple tags share names, the system resolves ambiguity through relationship analysis. A "Phoenix" tag might represent a city under "Geography" or a mythical creature

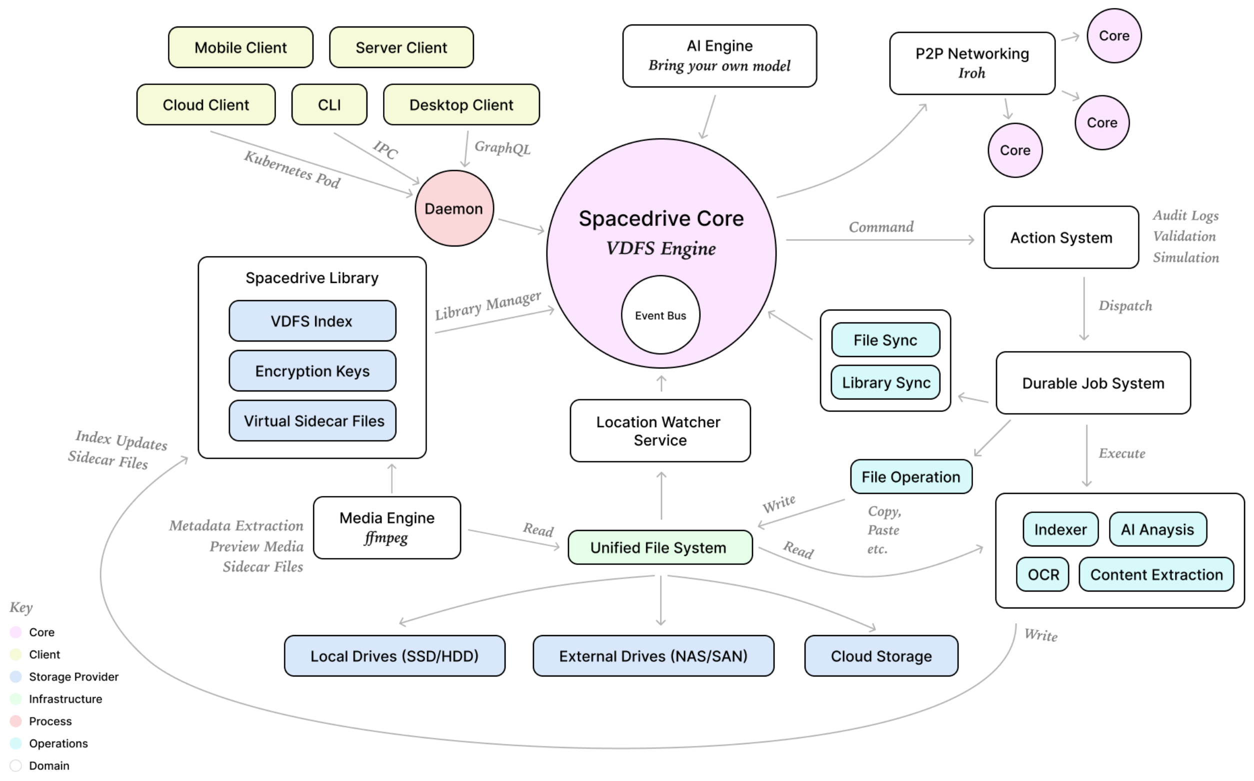


Figure 1: Spacedrive System Architecture

The provided figure illustrates the Spacedrive system architecture, a modular design centered around the **Spacedrive Core (VDFS Engine)**. A user is looking at the flow of a command as it originates from clients or automated systems like the AI Engine. The command is processed by the **Transactional Action System** for validation, then dispatched to the **Durable Job System** for execution. These jobs, which range from content indexing to file synchronization, interact with data via a **Unified File System** that sits above physical storage. The final state, including the file index and any extracted metadata, is maintained in the **Spacedrive Library**, which serves as the persistent state of the system.

under "Mythology", with automatic contextual display based on the tag's position in the semantic graph.

Organizational Benefits:

- **Implicit Classification**: Tagging a document with "Quarterly Report" automatically inherits organizational context from "Business Documents" and "Financial Records"
- **Semantic Discovery**: Queries for "Corporate Materials" surface all descendant content through graph traversal
- **Emergent Patterns**: The system surfaces inferred organizational connections

Advanced Tag Capabilities

Beyond basic labeling, tags function as rich metadata objects:

- **Organizational Roles**: Tags marked as organizational anchors create visual hierarchies in the interface
- **Privacy Controls**: Archive-style tags can shield content from standard searches while maintaining accessibility
- **Visual Semantics**: Customizable appearance properties encode meaning through color and iconography
- **Compositional Attributes**: Support attribute composition (e.g., "Technical Document" WITH "Confidential" AND "2024 Q3")

This architecture treats tags as first-class metadata that capture relationships in personal data organization, scaling from basic keyword tagging to organizational knowledge management.

4.1.4 SdPath: Universal File Addressing. Central to the VDFS abstraction is **SdPath**—a universal addressing system that makes device boundaries transparent. While the primary form provides a direct physical coordinate (device identifier + local path), Spacedrive also supports a content-aware addressing mode that transforms SdPath from a simple pointer into a content resolver.

```rust
#[derive(Clone, Debug)]
pub enum SdPath {
    // Physical addressing: device + path
    Physical {
        device_id: DeviceId,
        local_path: PathBuf
    },
    // Cloud addressing: cloud volume + path
    Cloud {
        volume_id: Uuid,
        path: String  // Cloud-native path format
    },
    // Content-aware addressing: find optimal instance
    Content {
        content_id: Uuid
    },
}
```

Listing 1: SdPath supports both physical and content-aware addressing

Content-Aware Addressing and Optimal Path Resolution

Content-aware addressing allows Spacedrive to automatically find the best available copy of a file across all devices. Instead of failing when a specific device is offline, the system automatically locates and uses alternative copies—turning device-specific file paths into content handles that continue to function.

When an operation is initiated with a content-aware `SdPath`, Spacedrive performs an **optimal path resolution** query against the Library index:

(1) **Content Lookup**: Query the `Content Identity` table to find all instances of the file content across all devices
(2) **Candidate Evaluation**: Evaluate each instance based on a cost function considering:
    - **Locality**: Local device copies prioritized above all others
    - **Network Proximity**: Iroh provides real-time latency and bandwidth estimates
    - **Device Availability**: Filter for currently online devices
    - **Storage Tier**: Volume-Aware Storage Foundation prioritizes SSD over HDD
(3) **Path Selection**: Select the lowest-cost valid path and proceed transparently

This mechanism makes file operations more resilient. If a user requests a file from an offline laptop, Spacedrive can source the identical content from a NAS on the local network. This elevates `SdPath` from a simple address to an abstract, location-independent handle for content.

Practical Applications

This addressing system enables operations that were previously impossible or extremely complex:

- **Availability-Aware Operations**: File operations can succeed even when the original source is offline
- **Performance-Aware Source Selection**: Automatically select the fastest available source
- **Simplified Development**: Applications reference content by ID without managing device availability
- **Automatic Failover**: Operations automatically switch to alternative sources

This abstraction simplifies cross-device operations into type-safe function calls. The distributed filesystem appears local to users and developers.

Universal URI Scheme. To make these addressing modes accessible and ergonomic for clients, APIs, and command-line interfaces, Spacedrive establishes a unified string-based URI scheme. This allows any path, whether physical or content-based, to be represented as a simple, standardized string:

- **Physical Path**: A physical location is represented as `sd://<device_id>/path/to/file`.
- **Cloud Path**: A cloud volume location is represented as `sd://cloud/<volume_id>/path/to/file`.
- **Content Path**: A content-aware handle is represented as `sd://content/<content_id>`.

This design decouples clients from the complexity of path resolution. A user interface or script can operate entirely on these URI strings, passing them to the core engine, which is then responsible for parsing the URI and dispatching the appropriate resolution logic.

4.1.5 *The Virtual Sidecar System: Managing Derivative Data.*

Spacedrive's VDFS model extends beyond managing original user files to also include first-class support for derivative data through a Virtual Sidecar System. For any given Entry, Spacedrive can create and manage a set of associated files—such as thumbnails, OCR text data, video transcripts, or transcoded media proxies—without ever modifying the original file.

These sidecar files are stored within a managed directory inside the portable `.sdlibrary` container and are linked to the original Entry via its globally unique ID in the VDFS index. This offers several key advantages:

- **Integrity**: The user's original files are never altered, preserving their integrity and original metadata.
- **Portability**: All AI-generated metadata and other derivative data travels with the Library, making the entire organized ecosystem portable.
- **Decoupling**: Analysis-extraction processes are decoupled from core indexing. New analysis capabilities (e.g., new AI models) can be added in the future and run on existing files to generate new sidecars without re-indexing the entire library.

This system supports Spacedrive's derived-data features, providing inputs for semantic search and the AI layer.

4.1.6 *Advanced File Type System.*

Spacedrive implements a multi-method file type identification system that goes beyond traditional extension-based detection to provide semantic categorization and accurate content identification.

Multi-Method Identification. The system combines multiple detection strategies to improve accuracy:

- **Extension Matching**: Fast initial identification with priority-based conflict resolution (e.g., `.ts` files prioritize TypeScript over MPEG-TS based on context)
- **Magic Byte Detection**: Binary pattern matching at specific offsets for definitive file type verification
- **Content Analysis**: Heuristic analysis for text and code files when other methods are ambiguous
- **Confidence Scoring**: Each identification method provides confidence levels (60-100%) enabling fallbacks

Semantic Categorization. Files are automatically grouped into semantic categories (ContentKind) that enable intuitive organization and specialized handling:

- **Media Types**: Image, Video, Audio—enabling gallery views and media players
- **Document Types**: Document, Book—for reading interfaces and text extraction
- **Development**: Code, Config, Database—supporting syntax highlighting and project views

- **System Files**: Executable, Binary, Archive—with appropriate security warnings
- **Specialized**: Mesh (3D models), Font, Encrypted, Key—each with tailored handling

Extensibility. New file types are defined through declarative TOML specifications:

```toml
[[file_types]]
id = "image/avif"
name = "AV1 Image File"
extensions = ["avif", "avifs"]
mime_types = ["image/avif", "image/avif-sequence"]
category = "image"
priority = 95

[[file_types.magic_bytes]]
pattern = "00 00 00 ?? 66 74 79 70 61 76 69 66"
offset = 0
priority = 100

[file_types.metadata]
supports_transparency = true
supports_animation = true
codec = "av1"
```

Listing 2: Example file type definition for AVIF images

The magic byte system supports detailed patterns:

- Exact bytes: "FF D8" for JPEG headers
- Wildcards: "52 49 46 46 ?? ?? ?? ?? 57 45 42 50" for WebP
- Ranges: "00-1F" for any control character
- Multiple patterns with different offsets and priorities

4.2 Content Identity: Deduplication and Redundancy

Spacedrive's content-addressable storage system serves a dual purpose: it reduces storage usage through content-aware deduplication [?] and tracks redundancy across devices. This unified approach turns content identification into a practical data protection approach:

4.2.1 Adaptive Hashing Strategy. The system employs different strategies based on file size, inspired by systems like LBFS [?] that demonstrated the benefits of content-aware chunking:

Adaptive Content Fingerprinting Strategy

Spacedrive uses a size-based approach to create unique fingerprints for files:

Small Files (under 100KB):

- Complete content analysis for exact matching of identical files
- Detects identical files via full-hash comparison

Large Files (over 100KB):

- Strategic sampling from beginning, middle, and end segments
- Maintains deduplication effectiveness while preserving real-time performance

Empty Files:

- Ignored and do not receive a content identity

This approach enables effective deduplication on consumer hardware—recognizing that `vacation_video.mp4` on your laptop is identical to `backup_copy.mp4` on your external drive, even with different names and locations.

The hashing function, $H(x)$, for a file F of size S_F can be more accurately represented as:

$$H_{adaptive}(F) = \begin{cases} H(size \oplus F) & \text{if } S_F < \\ H(size \oplus H_{header} \oplus S_1 \oplus S_2 \oplus S_3 \oplus S_4 \oplus F_{footer}) & \text{if } S_F \geq \end{cases}$$

Where $H(x)$ is the BLAKE3 hash function including the file size in bytes as a prefix, H_{header} is the 8KB header, S_1 through S_4 are four 10KB samples evenly distributed through the file body, F_{footer} is the 8KB footer, and $\oplus$ denotes the concatenation operation.

Two-Tier Hashing: Performance vs. Security Trade-off. The Content Identity system separates identity (for deduplication) from integrity (for verification) through a two-tier hashing architecture:

Initial Indexing (Sampled Hash): During the content indexing phase, only the sampled hash is computed. For large files, this reads 58KB of data instead of the entire file. A 10GB video requires 58KB of I/O instead of 10GB—approximately 170,000× less data read. Only the first 16 characters of the resulting BLAKE3 hex hash are stored as the content identifier, providing sufficient uniqueness for general deduplication.

Lazy Integrity Validation: Complete BLAKE3 hashes of entire file contents are generated on-demand by background `ValidationJobs`:

- **Scheduling**: ValidationJobs run during idle periods (low CPU usage, device charging)
- **Priority**: Security-sensitive files and recently transferred files validated first
- **Verification**: Full hash compared against expected content ID; mismatches trigger corruption alerts
- **Recovery**: System automatically offers restoration from known-good copies on other devices

When Full Integrity Matters:

- **File Transfers**: Verify no corruption during cross-device operations
- **Backup Verification**: Ensure bit-perfect copies in archival storage
- **Security-Sensitive Files**: Cryptographic proof of content authenticity
- **Forensic Analysis**: Detect tampering in legal or compliance scenarios

This enables accelerated indexing while asynchronously preserving cryptographic guarantees when full integrity matters. Users benefit from immediate deduplication without waiting hours for full cryptographic hashing of large media libraries.

4.2.2 Content Identity Management. Each unique piece of content receives a **Content Identity** record that tracks all instances across the Library:

Content Identity Tracking

Each unique piece of content receives a detailed identity record:

Users can execute expressive queries like "show all instances of this photo across devices" and "calculate storage savings from deduplication." The system recognizes that `/Users/alice/vacation.jpg` and `/backup/IMG_1234.jpg` contain identical content, presenting a unified view while maintaining the actual filesystem locations.

Property	Description
Unique ID	Permanent content identifier
Content Hash	Fast sampled hash (first 16 chars, e.g., "a1b2c3d4e5f6g7h8")
Integrity Hash	Full BLAKE3 hash for validation (generated lazily)
Content Type	Classification (Image, Video, etc.)
Instance Count	Copies across all devices
Total Size	Storage per copy
Timeline	First found/last verified

Table 3: Content identity tracking

4.2.3 Redundancy Analysis and Protection. Beyond deduplication, the Content Identity system adds redundancy analysis to support data protection:

At Risk Files: For any file or folder, the system knows how many copies exist and where they're located. A query might reveal: "Your wedding photos have 3 copies: MacBook Pro (SSD), Home NAS (RAID), and Cloud Backup." By intersecting two arbitrary locations, a user can instantly identify files missing from backups. Combining this redundancy data with volume classifications, Spacedrive identifies at-risk files—critical documents with only one copy on a laptop SSD are flagged as high-risk, while files with copies on both primary and backup storage are marked as secure. These signals inform AI-generated protection suggestions.

Integrity Verification: The system employs a two-tier hashing strategy for optimal performance. During initial indexing, only the fast content hash (sampled for large files) is generated. Full integrity verification happens lazily through background `ValidationJobs` that generate complete BLAKE3 hashes of entire file contents. This approach allows rapid indexing while still providing cryptographic integrity guarantees when needed. Any mismatch between a file's integrity hash and its actual content indicates potential corruption, triggering immediate alerts and offering restoration from known-good copies on other devices.

Protection Suggestions: When the AI identifies important files lacking redundancy—such as recently imported photos or new project documents—it generates actionable suggestions: "I noticed your 'Tax Documents 2024' folder exists only on your laptop. Would you like to create a backup on your NAS?" These suggestions appear as pre-visualized Actions, showing exactly what will happen.

4.3 The Indexing Engine

Key Takeaways

- **Five-Phase Pipeline**: Discovery → Processing → Aggregation → Content Identification → Finalizing
- **Hybrid Architecture**: Dual-layer system with ephemeral (RAM) and persistent (SQLite) indexes
- **Memory Optimizations**: NodeArena slab allocator and NameCache string interning (50 bytes per entry)
- **Real-Time Monitoring**: Platform-native watchers with unified ChangeHandler interface
- **Remote Volume Support**: Extends to clouds/protocols via OpenDAL with ranged reads for efficiency

The Spacedrive index is the cornerstone of the VDFS, a multi-phase directory-traversal system designed for performance, fault tolerance, and flexibility on consumer hardware, with support for both local file systems and remote volumes through OpenDAL integration.

4.3.1 Multi-Phase Processing Pipeline. To manage the complexity of file system analysis, the indexer employs a five-phase pipeline. The ephemeral engine runs only Phase 1 (Discovery) for instant in-memory browsing, while the persistent engine executes all five phases with full database writes and content analysis.

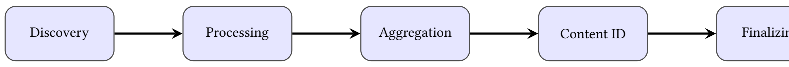


Figure 2: The five phases of the Spacedrive indexing pipeline. The ephemeral engine runs only Phase 1 for instant in-memory browsing, while the persistent engine executes all five phases.

- **Discovery Phase**: The engine performs a recursive traversal of a `Location`'s file system. It applies a set of predefined filter rules to ignore system and hidden files, caches, and development directories (e.g., `.git`, `node_modules`). During traversal, the system captures filesystem metadata including inodes and modification timestamps for each entry. Discovered items are collected into batches for efficient processing.
- **Processing Phase**: Each batch of discovered entries is processed to create or update records in the database. This phase includes **change detection**, which uses the captured inode and modification timestamp data to identify new, modified, or moved files, ensuring that only necessary database updates are performed.
- **Aggregation Phase**: For directories, the engine performs a bottom-up traversal to calculate aggregate statistics, such as total size and file counts. This pre-calculation makes directory size lookups an O(1) operation.
- **Content Identification Phase**: For files, this phase generates a content hash (CAS ID—Content-Addressed Storage Identifier) for deduplication using BLAKE3. The system creates globally deterministic v5 UUIDs from content hashes, enabling offline duplicate detection across all devices without coordination. This phase also performs file type detection using a combination of extension matching and magic byte analysis through the FileTypeRegistry. For media files, the system leverages bundled FFmpeg libraries to extract rich metadata including duration, codec information, bitrate, resolution, and frame rate—all stored in dedicated media data tables for efficient querying.
- **Finalizing Phase**: This phase completes post-processing tasks including final directory aggregate updates and processor dispatch. In Deep Mode, this phase triggers thumbnail generation, OCR processing, and other analysis jobs defined by installed extensions. The phase marks the indexing operation as complete and updates the location's last-indexed timestamp.

After a file's content and type are identified in Phase 4, the Finalizing phase can dispatch specialized, asynchronous jobs for deeper analysis. These jobs are registered and defined by installed extensions, allowing for a modular approach to intelligence. For example, a Photos extension might queue an `ImageAnalysisJob` for images, while a developer extension might trigger a `CodeAnalysisJob` for source files. This ensures that deeper analysis is only performed when relevant to the user's workflow and installed extensions.

Checkpoint Architecture and Resumability. The indexing engine's design enables robust resumability critical for mobile devices and large libraries where indexing can span hours and face multiple interruptions.

Checkpoint Mechanism: Jobs checkpoint automatically after each batch. State serialization uses MessagePack for compact binary encoding. Each checkpoint captures:

- Current phase identifier
- Batch cursor position
- Set of processed entry IDs
- Phase-specific state (e.g., discovered paths queue, aggregation stack)

Resumability Flow: When an indexing job is interrupted:

(1) Job interrupted (crash, user cancellation, device offline, low battery)
(2) Current state persisted to `jobs.db` with last completed phase
(3) On restart: Load serialized state from database
(4) Jump to last completed phase, skip already-processed entries
(5) Continue from checkpoint cursor position

This transforms hours-long indexing operations into resilient, interruptible workflows. A user can index a 500GB photo library over multiple days, pausing for battery conservation or device shutdown, with zero redundant work on resume.

Ephemeral Mode: The indexer's ephemeral mode creates in-memory `Entry` records for unindexed paths, enabling three critical use cases:

- **Exploratory Browsing**: Users can explore external drives or removable media before committing to permanent indexing, with full metadata capabilities (tagging, organizing) available immediately
- **Remote Filesystem Access**: When browsing a peer device, lightweight ephemeral entries stream in real-time without polluting the local database
- **Lazy Refresh Navigation**: During directory browsing, existing indexed data displays instantly while a background validation job updates changed entries

These in-memory ephemeral entries appear indistinguishable from persisted entries in the UI, creating a unified browsing experience across indexed and non-indexed data.

Memory Optimizations for Ephemeral Indexing. To make ephemeral mode practical for browsing millions of files, Spacedrive employs specialized data structures that reduce memory overhead from 250 bytes per entry (naive approach) to approximately 50 bytes per entry—a 5× reduction.

NodeArena Slab Allocator: Instead of storing entries in a `HashMap<PathBuf, Entry>` with 64-bit pointers, the system uses a contiguous memory slab indexed by 32-bit integers. This `NodeArena` allocates `FileNode` entries sequentially and maintains a free list for reusing deleted slots. The 32-bit node IDs reduce pointer overhead by 50% while improving cache locality through sequential memory layout.

NameCache String Interning: Filesystem names repeat heavily—`index.js`, `.DS_Store`, `package.json` appear thousands of times in typical projects. The `NameCache` stores each unique string once in an `Arc<str>` pool and references them by 32-bit IDs. For a Node.js project with 1,000 packages each containing `index.js`, string interning stores "index.js" once (8 bytes) plus 1,000 references (4 KB) instead of 1,000 copies (8 KB), achieving 50% reduction. Common names like `.git`, `README.md`, and system files deduplicate heavily across directory trees.

NameRegistry for Fast Lookups: A `BTreeMap<NameId, Vec<NodeId>` enables O(log n) "find by name" queries without full-text indexing. This allows instant lookups like "find all README.md files" by indexing on interned name IDs rather than string comparisons.

These optimizations enable browsing hundreds of thousands of files in ephemeral mode with under 50 MB of RAM, making Spacedrive viable as a daily-driver file manager for exploring external drives and network shares without database commits.

4.3.2 *Flexible Indexing Scopes and Persistence.*

A key innovation of the Spacedrive indexer is its ability to adapt to different use cases through two orthogonal dimensions: **scope** and **persistence mode**.

Indexing Scope: The indexer can perform either a full recursive scan of a directory tree or a shallow, single-level scan of immediate contents. The shallow mode is integral to Spacedrive's responsive UI navigation through a "lazy refresh" mechanism—when browsing directories, the system instantly presents existing indexed data while concurrently spawning a non-blocking validation job to ensure accuracy without sacrificing interactivity.

Persistence Mode: For managed `Locations`, indexing results are persisted to the Library's database. However, the indexer also supports an **ephemeral, in-memory mode**. First, users can browse external or temporary paths without polluting the main index; the system generates temporary `Entry` records on the fly, streaming them to the UI for responsive exploration. Second, this extends to **remote filesystems**—when a user browses a path on a paired device, Spacedrive initiates an ephemeral indexing job on the target device, streaming lightweight `Entry` records back in real time. These ephemeral entries, whether local or remote, are presented "inline" with persisted entries, creating a unified view of the distributed filesystem.

This flexibility is managed through a unified `IndexerJobConfig`, enabling fine-grained control from background library maintenance to real-time UI interactions.

4.3.3 *Locations and Real-Time Monitoring.*

A Spacedrive Library is composed of one or more **Locations**—managed directories that act as the entry points to a user's physical file systems. The **Location Watcher** service provides a cross-platform, real-time monitoring system that keeps the Spacedrive index synchronized with the underlying file system.

The Location as a Managed Entity

When a user adds a directory to a Spacedrive Library, it becomes a `Location`, a managed entity with its own configuration and lifecycle. Each Location provides granular control over how different parts of the user's data space are handled. Each `Location` has a specific **Index Mode** (`Shallow`, `Content`, or `Deep`), enabling users to apply different levels of analysis to different types of content (e.g., deep analysis for a photo library, shallow for a downloads folder).

The Location Watcher Service

The watcher service is the core of Spacedrive's real-time capabilities, providing an efficient file system monitoring solution.

Platform-Specific Optimizations

A key strength of the watcher is its use of platform-native APIs for optimal performance and reliability. This is a non-trivial engineering challenge, as each OS has unique behaviors.

- **macOS (`FSEvents`)**: The system correctly handles the ambiguous rename and move events from `FSEvents` by tracking file inodes to reliably link old and new paths.
- **Linux (`inotify`)**: The watcher leverages the efficiency of `inotify` for direct, recursive directory watching and uses cookie-based event correlation to reliably detect move operations.
- **Windows (`ReadDirectoryChangesW`)**: The implementation is designed to handle Windows-specific filesystem quirks, such as delayed file deletions caused by antivirus software or file locking. It does this by maintaining a "pending deletion" state to verify that a file is truly gone before emitting a deletion event.

Event Processing Pipeline

The watcher service is more than a simple event forwarder. It includes a processing pipeline:

- **Noise Filtering via IndexerRuler**: The watcher applies the same configurable **indexer rules engine** used during initial indexing through the `IndexerRuler` component. This unified filtering system provides toggleable system rules including `NO_HIDDEN` (skip dotfiles like `.git`, `.DS_Store`), `NO_DEV_DIRS` (skip `node_modules`, `target`, `dist`), `NO_SYSTEM` (skip OS folders like `System32`, `/proc`), and `NO_TEMP` (skip temporary files). When indexing inside Git repositories, the ruler automatically loads and applies `.gitignore` patterns, preventing build artifacts and local configuration from polluting the index. Rules are evaluated at the discovery edge—rejected files never enter the processing pipeline, providing 6-8× speedup on typical development directories by avoiding expensive database operations for thousands of dependency files.
- **Event Debouncing**: To prevent "event storms" during bulk operations (e.g., unzipping an archive), the system debounces file system events, consolidating rapid-fire changes into single, actionable events.
- **Responder Pipeline**: Processed events are dispatched to the **indexer responder**, which translates raw filesystem events into database operations. The responder resolves entry identities, updates metadata, triggers content processors (thumbnails, OCR, content hashing), and emits resource events that notify the sync service to propagate changes to other devices.

This real-time monitoring system helps Spacedrive maintain an up-to-date view of a user's files.

4.3.4 *Offline Recovery and Stale Detection.*

While the Location Watcher provides real-time monitoring during normal operation, a critical challenge arises when Spacedrive itself is offline—whether due to system shutdown, crashes, or disconnected storage volumes. When the system returns online, it must efficiently detect and reconcile any filesystem changes that occurred during its absence.

Design for Offline Window Tracking

The system architecture includes support for tracking watcher uptime and persisting last-shutdown timestamps per location. Upon startup, the planned behavior is to calculate the "offline window"—the period between the last shutdown and current time—defining the temporal scope for potential undetected changes. By comparing this offline period against filesystem modification times, the system can efficiently identify which portions of the filesystem require validation. Note that while the change detection infrastructure (ChangeDetector) and verification system (IndexVerifyAction) are implemented, *automatic startup stale detection is not yet fully deployed.* Currently, users can manually trigger verification via CLI commands when needed.

Stale Detection Algorithm

Rather than performing expensive full filesystem scans after every offline period, Spacedrive leverages a key property of modern filesystems: modification time propagation. On most operating systems (Windows NTFS, macOS APFS, and Linux ext4/btrfs), changes to files within nested directories update the modification timestamps of parent directories up the tree.

The stale detection algorithm operates as follows:

(1) Walk the directory tree starting from each Location root
(2) Compare directory modification times against the offline window
(3) If a directory's modification time falls within the offline window, mark it for deep scanning
(4) Recursively scan only marked directories and their contents
(5) Update the index with discovered changes while preserving unchanged portions

This approach substantially reduces re-indexing overhead. For example, in a Location with 100,000 files across 10,000 directories where only 50 files changed during offline time, the system might only need to deeply scan 200–300 directories rather than the entire tree.

Graceful Degradation for Unsupported Filesystems

For filesystems that don't reliably propagate modification times (such as certain network filesystems or FAT32), Spacedrive detects this limitation and falls back to an ephemeral deep indexing strategy. This approach leverages the existing multi-phase indexing pipeline but constrains execution to only the Discovery phase—rapidly traversing the filesystem to create lightweight Entry records without performing expensive content analysis. By comparing these ephemeral entries against the persisted database state, the system can efficiently identify additions, deletions, and modifications based on file metadata. Only the detected changes are then queued for full processing through the remaining indexing phases (content identification, media analysis, etc.). The system maintains filesystem capability profiles to automatically select the optimal detection strategy per Location.

This hybrid approach ensures Spacedrive maintains its performance advantages while guaranteeing index integrity, addressing a fundamental limitation of purely real-time monitoring systems.

4.3.5 Index Verification and Integrity Checking. To provide diagnostic capabilities and ensure index correctness, Spacedrive implements an `IndexVerifyAction` that performs on-demand integrity checks. This system runs a fresh ephemeral scan of a location and compares the in-memory results against the persistent database, identifying discrepancies without modifying any data.

The verification process detects three categories of integrity issues: **MissingFromIndex** (files exist on disk but not in database, indicating missed create events or indexing gaps), **StaleInIndex** (entries in database but files missing from filesystem, indicating missed delete events), and **MetadataMismatch** (size, modification time, or inode differences between database and filesystem, indicating missed modify events or filesystem corruption). Each issue includes the affected path, entry ID, and specific mismatch details.

The verification system generates detailed `IntegrityReport` outputs with per-file diagnostics and summary statistics, accessible via CLI commands or the UI. This is particularly valuable after extended offline periods, when debugging watcher issues, or before major library migrations. The ephemeral comparison approach—indexing into memory rather than database—ensures verification runs quickly (typically under 30 seconds for 100K files) and never corrupts the existing index. Users can trigger verification manually via `spacedrive verify <path>` or schedule periodic background checks for critical locations.

4.3.6 Extending the Indexer for Remote Volumes. Spacedrive's indexing pipeline extends naturally to remote volumes (e.g., cloud services like S3 or protocols like FTP) through integration with OpenDAL [?], a Rust-native library providing unified, asynchronous access to diverse storage backends. This allows treating remote storage as standard Locations without data duplication, leveraging ranged reads for efficiency and provider metadata for optimized hashing.

Key Adaptations:

- **Discovery Phase**: Use OpenDAL's `Lister` for streaming directory traversal, avoiding full loads for large remotes (e.g., petabyte S3 buckets).
- **Content Identification**: Prioritize provider-computed hashes where available (e.g., S3 ETag as MD5 for small files [?]; GCS MD5). Fallback to client-side adaptive hashing via ranged reads to sample chunks without full downloads.
- **Resilience**: Wrap operations in retries with exponential backoff to handle rate limits (e.g., S3's 3,500 requests/sec) and transient errors.
- **Metadata-Only Mode**: Initial indexing fetches only stats (size, mod time) for quick Entry creation; defer content analysis to on-demand jobs.

This design minimizes bandwidth and costs: hashing a 10GB file uses only 64KB of ranged data (8KB header + 4×10KB samples + 8KB footer) instead of full transfer. The implementation builds scheme-specific OpenDAL operators (for S3, FTP, etc.) from location configuration, retrieving credentials from the secure vault. Remote hashing applies the same adaptive BLAKE3 strategy as local files, using ranged reads for efficient sampling without downloading entire files. Recent OpenDAL enhancements (v0.49, 2025) improve concurrent listing and metadata handling [?].

4.3.7 High-Performance Hierarchical Indexing. To overcome the scaling limitations of traditional path-based queries, Spacedrive's index uses a **Closure Table** for all hierarchical data. This database pattern pre-calculates and stores all ancestor-descendant relationships in an `EntryClosure` table, transforming slow string comparisons into highly efficient, indexed integer joins.

When a new entry is created, its `parent_id` is used to transactionally populate the `entry_closure` table with all its ancestor relationships in a single, atomic operation. This keeps the hierarchy consistent.

A critical component of this system is the indexer's resilient change detection. To ensure data integrity, especially for offline changes, the indexer uses **inode tracking** to reliably differentiate between a file move and a delete/add operation. When a file is moved within the same volume, its inode remains constant. The indexer leverages this to identify the operation as a move, triggering a safe, transactional update of the entry's path and its position in the closure table, rather than performing a destructive and incorrect delete-and-re-add. This preserves all user metadata and ensures the integrity of the hierarchical index.

4.4 The Transactional Action System

Key Takeaways

- **New Paradigm**: Preview any operation before execution - see exactly what will happen
- **Durable Execution**: Operations become durable jobs that continue across device disconnections
- **Centralized Control**: All operations flow through type-safe action system with full audit logging

Traditional file management executes operations immediately without preview. Spacedrive's **Transactional Action System with Pre-visualization** treats operations as transactions that can be previewed.

This system allows any file system operation to be simulated in a "dry run" mode before execution. Powered by the VDFS library index, this simulation can pre-visualize the outcome of an action—including space savings, data deduplication, and the final state of all affected locations—without touching a single file.

4.4.1 The Action Lifecycle: Preview, Commit, Verify. Every action in Spacedrive follows a transactional lifecycle:

Intent & Preview: The user expresses an intent (e.g., "move photos from my phone to my NAS"). Spacedrive uses its index to generate a preview of the outcome. The system can accurately forecast the end state because it has a complete metadata map of all user data.

Commit: Once the user approves the preview, the action is committed to the Durable Job System. It becomes a resilient, resumable job that continues to execute when devices are online, even if network connectivity is intermittent.

Execution & Verification: The job is executed by the appropriate device agents when they come online. The system works to complete the job, verifying each step against the initial plan. This durability helps ensure that user intent is fulfilled without data loss or corruption.

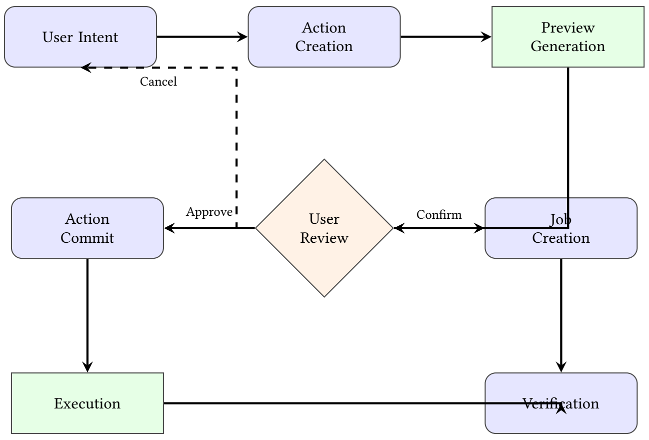
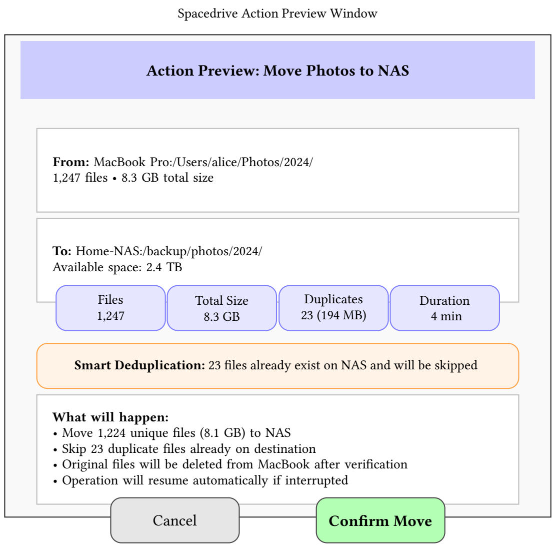


Figure 3: The Action Lifecycle: From user intent through preview, approval, and execution


Spacedrive Action Preview Window

Action Preview: Move Photos to NAS

From: MacBook Pro:/Users/alice/Photos/2024/
1,247 files • 8.3 GB total size

To: Home-NAS:/backup/photos/2024/
Available space: 2.4 TB

Files	Total Size	Duplicates	Duration
1,247	8.3 GB	23 (194 MB)	4 min

Smart Deduplication: 23 files already exist on NAS and will be skipped

What will happen:
- Move 1,224 unique files (8.1 GB) to NAS
- Skip 23 duplicate files already on destination
- Original files will be deleted from MacBook after verification
- Operation will resume automatically if interrupted

Figure 4: Action Preview UI Mockup: Users see exactly what will happen before any files are modified, including deduplication savings and potential issues.

Accessibility Integration. The UI supports screen readers for natural language commands and high-contrast modes for previews, ensuring inclusive AI access [?].

4.4.2 *The Simulation Engine.*

The Spacedrive index serves as the basis for a simulation engine that provides accurate operation previews through complete knowledge of the VDFS state.

Index-Based Simulation Architecture. The simulation engine's power derives from having complete metadata for every file without filesystem access:

- **Zero I/O Preview**: All predictions via SQL queries against the VDFS index—no filesystem reads during preview generation
- **Complete Knowledge**: Every file's size, content hash, location, permissions, and relationships are instantly available
- **Deduplication Awareness**: Content Identity table enables instant duplicate detection across all devices
- **Hierarchical Operations**: Closure table enables O(1) queries for directory subtrees, avoiding recursive scans

This architecture transforms operation planning from a slow, I/O-bound process into a fast, in-memory query execution.

Content-Aware Path Resolution. For operations using `SdPath::Content` addressing, the simulation engine performs optimal path resolution through a cost function evaluation:

1. **Content Lookup**: Query Content Identity table for all file instances across all devices
2. **Candidate Evaluation**: Score each instance using a cost function:
   - **Locality**: Local device copies = 0 cost (instant access)
   - **Network Proximity**: Iroh provides real-time latency/bandwidth estimates per peer
   - **Storage Tier**: PhysicalClass prioritizes SSD over HDD for source files
   - **Device Availability**: Filter for currently online devices only
3. **Path Selection**: Select lowest-cost valid path; simulation proceeds with this resolved path

This enables accurate time estimation even when the optimal source device differs from the user's assumption.

Operational Conflict Detection

The closure table also significantly accelerates the preview generation itself. When simulating a move or copy of a large directory, the system can determine the entire scope of the operation (all descendant files and folders) with a single, efficient query, rather than a slow recursive path scan. The simulation engine identifies operational conflicts that would cause traditional file operations to fail:

- **Storage Constraints**: Calculates exact space requirements and verifies availability on target devices
- **Permission Violations**: Detects write-protected locations or access-restricted files before attempting operations
- **Path Conflicts**: Identifies naming collisions and circular reference issues in complex move operations
- **Resource Limitations**: Estimates memory and bandwidth requirements against device capabilities

The simulation engine catches operational conflicts during planning, while Library Sync's per-model resolution handles the rare synchronization conflicts from concurrent modifications (Section 4.5.1). This dual approach improves reliability in distributed file management.

This is particularly powerful when combined with Native Storage Tiering (Section 4.8). If a user attempts an operation on a Location they have marked with a `Hot` logical class, but it resides on a Volume physically classified as `Cold`, the simulation engine generates a clear warning in the preview: "**Warning:** This operation targets a 'hot' location on a slow archive drive. Estimated completion time may be longer than expected."

Time Estimation. The Simulation Engine combines multiple data sources to provide accurate operation time estimates:

- **Volume Performance Metrics**: Real-time read/write speeds from continuous monitoring
- **Network Conditions**: Current bandwidth and latency from Iroh's measurements
- **Historical Data**: Previous operations on similar files and paths
- **Operation Complexity**: Number of files, total size, and fragmentation
- **Storage Type Awareness**: Different strategies for local vs cloud storage

For example, when copying 10GB across devices, the estimation considers:

- Source volume read speed: 250 MB/s (measured)
- Network throughput: 45 MB/s (current P2P bandwidth)
- Destination write speed: 180 MB/s (measured)
- Bottleneck: Network at 45 MB/s
- Estimated time: 3 minutes 45 seconds (with 10% buffer)

For cloud operations, additional factors apply:

- API rate limits (e.g., 1000 requests/second for S3)
- Chunk size optimization (balancing throughput vs memory)
- Parallel stream count (typically 4-8 for cloud providers)
- Resume capability for long-running transfers

This transparency helps users make informed decisions about when and how to execute operations, especially for large-scale cloud migrations.

4.4.3 *Centralized Operation Control.*

Rather than allowing direct operation dispatch throughout the codebase, Spacedrive routes all user actions through a centralized **Action System** that provides consistent validation, execution, and logging:

The Action System employs a centralized enumeration that captures every possible user operation, distinguishing between global actions (system-level operations like library creation) and library-scoped actions (operations within a specific Library context). This design provides clear authorization boundaries and enables complete tracking of all user-initiated operations.

4.4.4 *Type-Safe Action Construction.*

The system employs a **builder pattern** for type-safe action construction that integrates with CLI and API inputs:

```rust
// Builder pattern ensures valid action construction
let action = FileCopyAction::builder()
    .source_paths(vec!["/docs/report.pdf", "/docs/data.
    csv"])
    .target_path("/backup/2024/")
    .mode(TransferMode::Move)  // Move instead of copy
    .verify_checksum(true)      // Ensure integrity
    .preserve_timestamps(true)   // Keep original dates
    .on_conflict(ConflictStrategy::Skip)
    .build()?;  // Returns error if invalid combination

// Actions are serializable for durability
let job = Job::from_action(action);
queue.push(job).await?;
```

Listing 3: Type-safe action construction with builder pattern

This builder approach provides **compile-time validation** of action parameters, preventing invalid operations from reaching the execution layer while maintaining ergonomic APIs for both programmatic and command-line usage.

4.4.5 *Complete Audit Logging.*

Every library-scoped action automatically receives full audit logging through the database layer:

Audit Field	Information Captured
Action Type	Operation performed (e.g., "file.copy")
Device	Initiating device identifier
Resources	Files/folders/locations affected
Status	Previewed → Committed → Complete
Job Link	Background job reference
Timing	Start/end times, duration
Errors	Failure details if applicable
Results	Outcome and metrics

Table 4: Detailed audit trail fields

The **ActionManager** automatically creates audit entries for both preview and execution phases, tracking the complete action lifecycle from initial intent through final completion.

4.4.6 *Dynamic Handler Registry.*

The Action System employs a dynamic registry pattern using Rust's **inventory** crate for automatic handler discovery:

Extensible Action Handler System

Spacedrive's Action System uses a self-registering architecture that automatically discovers available operations:

- **Automatic Discovery**: New operations register themselves when added to the codebase
- **No Central Maintenance**: Adding new file operations requires no manual registry updates
- **Type Safety**: Each operation handler is validated at compile time
- **Consistent Interface**: All operations (file copy, location management, etc.) follow the same patterns

This enables easy extension of Spacedrive's capabilities while maintaining system reliability and consistent user experience across all operations.

4.4.7 *Foundation for Advanced Capabilities.*

The Action System's centralized architecture enables advanced features that would be difficult to implement across a distributed codebase:

Enterprise-Grade RBAC Foundation

The centralized Action System is architected as the foundation for granular Role-Based Access Control (RBAC), essential for team collaboration and enterprise deployment:

- **Role Definitions**: Standard roles like "Viewer" (read-only), "Contributor" (read/write), "Manager" (full control), and custom roles tailored to organizational needs
- **Granular Permissions**: Action-level control enabling scenarios like "can upload but not delete" or "can tag but not move files"

- **Location-Based Access**: Restrict access to specific Locations or paths (e.g., "Finance team accesses /financial-data, Creative team accesses /assets")
- **Inheritance and Groups**: Permission inheritance through organizational groups with override capabilities
- **Temporal Controls**: Time-based access for contractors or temporary project members
- **Audit Trail Integration**: Every permission check logged with full context for compliance and security reviews

Undo Capabilities

The detailed audit trail provides the foundation for reliable operation reversal:

- **Safe Undo Logic**: System understands how to safely reverse each operation type
- **Dependency Tracking**: Prevents undoing operations that other actions depend on
- **Selective Reversal**: Undo specific parts of complex operations (e.g., "undo copying just these 3 files")
- **Cross-Device Coordination**: Undo operations that span multiple devices with proper cleanup

4.4.8 Remote Action Dispatch. A key benefit of combining the Action System with the `SdPath` universal addressing scheme is the ability to dispatch actions where the execution occurs transparently on a remote device. For example, a `FileCopyAction` can specify a source `SdPath` on Device A and a destination `SdPath` on Device B. When this action is committed, the Job System routes the work to the appropriate device. It creates a sender job on Device A and a corresponding receiver job on Device B, which coordinate the transfer over the secure P2P network. This architecture makes complex cross-device workflows trivial to express and execute, as the underlying network communication and state management are handled automatically by the core engine. All remote operations are still subject to the same validation, preview, and audit logging as local actions, ensuring a consistent security model across the entire VDFS.

4.4.9 Handling File Ingestion and Uploads. While much of Spacedrive's power comes from indexing existing files, a critical function is the ingestion of new files from external sources, such as a web interface or a drag-and-drop operation into the application. This process is managed through a specialized **Ingestion Workflow**, built upon the Transactional Action System.

At the heart of this workflow is the concept of a user-configurable **Ingest Location**, colloquially known as an "Inbox" or "quarantine zone." This is a designated default directory on a preferred, often always-online, device (such as a home server or a Cloud Core instance). When a user uploads a file without specifying an explicit destination:

(1) A `FileIngestAction` is created. The source is the uploaded data stream, and the destination defaults to the primary Ingest Location.
(2) The Action System routes this job to the destination device, which handles the secure file transfer via the Iroh protocol.
(3) Upon successful transfer, the file is written to the Ingest Location, and a new `Entry` is created in the VDFS index.

This architecture ensures that new files are added to the user's data space in a transactional, reliable manner. Once the file becomes a managed `Entry`, it is immediately available for the AI layer to analyze and organize, as detailed in Section 4.6.4.

4.5 Library Sync and Networking

Spacedrive's distributed architecture requires synchronization and networking capable of maintaining consistency across devices while preserving local-first semantics. This section presents the approach to synchronization through domain separation and the Iroh-powered networking infrastructure [?] that enables peer-to-peer communication.

4.5.1 Library Sync via Domain Separation. Traditional distributed consensus algorithms struggle with the mixed requirements of personal data management. Spacedrive's Library Sync architecture sidesteps this complexity through a leaderless, peer-to-peer hybrid model that aligns synchronization strategy with data ownership. The key insight is that personal file management exhibits a natural ownership hierarchy: each device has exclusive authority over its local filesystem, while user-generated metadata (tags, collections) represents truly shared state that any device may modify.

This domain separation eliminates central coordinators, single points of failure, and the need for leader election. Data falls into two categories with tailored replication strategies.

Device-Authoritative Data. Each device maintains authoritative control over its own filesystem index. Ownership flows through volumes: each device owns its attached volumes, and locations and entries inherit ownership from their volume. This indirection enables portable storage—when an external drive moves between machines, a single volume update transfers ownership of all associated data instantly. Since no other device can modify another's local filesystem, write conflicts for this data class are impossible by construction. Synchronization employs a per-resource watermark system that enables efficient incremental sync.

Rather than tracking sync progress globally, the system maintains independent watermarks for each resource type (locations, entries, volumes). This granular approach enables targeted recovery: if entry synchronization falls behind, only entries require resyncing while location and volume sync remain unaffected. Each watermark advances only when data is actually received—a critical invariant that prevents data loss when network requests return empty results.

When devices reconnect after an offline period, incremental catch-up requests only records newer than the stored watermark. The response uses cursor-based pagination with a `timestamp|uuid` format to handle large result sets and ensure no records are skipped even when timestamps collide. If a watermark exceeds a configurable age threshold (default 25 days), the system forces a full sync to ensure consistency, as tombstones for deletions may have been pruned beyond that point.

Shared Data. User-generated metadata such as tags and collections is not owned by any single device and can be modified concurrently. This domain uses a lightweight, log-based approach where each device maintains a small per-device log (`sync.db`) of changes to shared resources. Each entry carries a Hybrid Logical Clock (HLC) timestamp—a distributed timestamp combining physical time with a logical counter that provides total ordering across all devices without clock synchronization.

The HLC structure encodes timestamp, counter, and device identifier:

$$\text{HLC} = (\text{timestamp}_{64}, \text{counter}_{64}, \text{device\_id}_{128}) \qquad (1)$$

Comparison proceeds lexicographically: first by timestamp, then counter, then device ID. This guarantees a total ordering over all events in the system, enabling deterministic conflict resolution regardless of the order in which devices receive updates.

Conflict resolution is per-model. Tag assignments use implicit union merge—when two devices tag the same file with different tags, both assignments sync as separate records and the file ends up with all tags. Scalar values such as favorites use last-writer-wins based on HLC ordering. These logs are aggressively pruned once all peers acknowledge receipt, typically maintaining fewer than 1,000 entries even with active multi-device usage.

Explicit File Transfer. Unlike automatic synchronization, file movements and copying in Spacedrive are explicit user commands. When a user requests "copy my photos to the backup drive," this creates a deliberate operation with clear source and destination specifications, predictable error handling with retry logic, and progress tracking. This separation eliminates the complexity of implicit file content synchronization, treating transfers as user-initiated operations with well-defined semantics.

The leaderless architecture provides several practical advantages. Any device can make changes without waiting for a coordinator. Devices queue changes locally and sync when connectivity returns, enabling true offline-first operation without read-only restrictions. No single point of failure exists—any peer can provide synchronization state to new devices. The absence of leader election, heartbeat mechanisms, and failover logic results in approximately 800 fewer lines of coordination code compared to leader-based alternatives.

4.5.2 Iroh-Powered Network Infrastructure. Spacedrive's networking architecture consolidates sync, file transfer, and device discovery into a unified layer powered by Iroh. Rather than managing separate networking stacks for each protocol, all communication flows through a single QUIC endpoint per device.

Protocol Multiplexing. Application-Layer Protocol Negotiation (ALPN) enables multiple Spacedrive protocols to coexist on a single QUIC connection. One connection per device pair handles pairing, sync, file transfer, and messaging, with each protocol identified by its ALPN string (e.g., `spacedrive/sync/1`). Protocols become streams rather than separate connections, sharing congestion control and eliminating coordination overhead. This consolidation reduces connection establishment time to under 2 seconds (from 3–5 seconds with separate connections) and eliminates approximately 60% of networking coordination code.

To prevent race conditions inherent in peer-to-peer networks, Spacedrive employs deterministic connection roles: only the device with the lexicographically lower NodeId initiates outbound connections. This eliminates simultaneous connection attempts and simplifies the connection state machine, as each device knows its role (initiator or responder) deterministically and consistently across reconnection attempts.

Pairing and Security. Initial device pairing employs a challenge-response protocol built on BIP39 mnemonic codes—12-word phrases derived from 256-bit secrets that provide human-readable verification. The four-message handshake prevents replay attacks, while Ed25519 signatures provide cryptographic authentication against man-in-the-middle attacks. Visual device fingerprints offer an additional verification layer during initial pairing.

The underlying QUIC transport provides ChaCha20-Poly1305 encryption, stream multiplexing for concurrent operations, BBR congestion control, and zero round-trip connection resumption. All traffic is encrypted end-to-end regardless of the network path.

Network Discovery and Relay. Devices discover each other through multiple channels simultaneously: local network mDNS broadcasting, DNS-based discovery for known devices, and relay server coordination for NAT traversal. When direct connection is not possible due to firewalls or symmetric NAT, the system routes through relay servers while maintaining end-to-end encryption—relays see only encrypted traffic and cannot inspect content.

While Spacedrive provides public relay servers, the architecture supports self-hosted deployments for organizations requiring data sovereignty or operating air-gapped networks. The relay service deploys as a single binary with minimal configuration, enabling geographic distribution near users for optimal latency. This flexibility accommodates enterprises with data residency requirements and organizations building private overlay networks.

Network Topologies. The Iroh-based networking supports multiple deployment topologies:

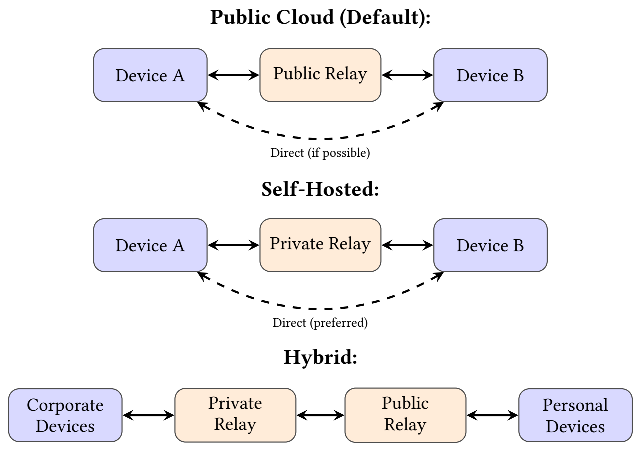


Figure 5: Network Architecture Flexibility: Spacedrive supports multiple network topologies to adapt to different deployment scenarios.

4.5.3 Spacedrop: Ephemeral Secure Sharing. Beyond trusted device pairing, Spacedrive implements Spacedrop—an ephemeral file sharing protocol enabling secure transfers between devices without prior relationships. Each session uses ephemeral ECDH key exchange, providing perfect forward secrecy: compromising device keys cannot decrypt past transfers, as fresh ephemeral keys are generated per session and discarded after completion.

Unlike automatic transfers between paired devices, every Spacedrop requires explicit receiver acceptance. The receiver sees the sender's device name, file metadata, and optional message before accepting, maintaining user control over incoming data. Discovery adapts to available connectivity: mDNS broadcasts for local network discovery, optional Bluetooth Low Energy advertisements for proximity detection without shared Wi-Fi, and DHT fallback for internet-wide discovery when local mechanisms fail.

Spacedrop can optionally leverage relay nodes to enable asynchronous transfers. Users "drop" files to a relay and receive a shareable link, allowing recipients to download later—combining end-to-end encryption with the convenience of hosted file sharing services.

4.5.4 Sync Conduits: Index-Based Content Synchronization. While Library Sync manages replication of the VDFS index and metadata, physical synchronization of file content is handled by Sync Conduits—durable, configurable jobs that define content flow between source and destination entries. This explicit, user-controlled approach ensures transparent control over data movement.

Sync Conduits leverage the fully replicated VDFS index to perform sync operations through database queries rather than filesystem scanning. Since Library Sync ensures all devices maintain a consistent view of filesystem state, determining what needs synchronization becomes a straightforward SQL query comparing content hashes and timestamps at source and destination locations. The system verifies Library Sync completion before execution, then dispatches `FileCopyJob` and `DeleteJob` instances using the proven job system infrastructure. The destination device validates incoming files against content hashes, preventing duplicate transfers.

Four sync modes accommodate different use cases. **Mirror** creates an exact one-way copy, propagating creates, modifies, and deletes for backup scenarios. **Bidirectional** keeps two locations identical with changes flowing both directions, using last-write-wins based on index timestamps. **Selective** manages limited local storage by keeping frequently accessed files local while moving older content to secondary storage, with files remaining visible in the index for on-demand retrieval. **Archive** moves files to long-term storage, removing sources only after cryptographic verification via content hash comparison.

The index-based architecture provides performance advantages over traditional sync: no directory traversal is needed, files excluded from indexing are automatically excluded from sync, and state tracking uses index timestamps rather than hidden sync logs or state files.

4.6 AI-Native VDFS: From Semantic Search to Intelligent Management

Key Takeaways

- **Extension-Based Agents**: AI agents are implemented as sandboxed WASM extensions with domain-specific knowledge and persistent memory
- **Event-Driven Intelligence**: Agents react to filesystem events, scheduled triggers, and user requests through a unified event system
- **Privacy-First AI**: All agent processing runs locally within secure WASM sandboxes, with fine-grained permission controls

While many systems treat AI as an additive feature, Spacedrive is architected as an **AI-native data space** where specialized intelligence is deeply integrated through its extension architecture. Rather than a single monolithic AI assistant, Spacedrive enables domain-specific agents—such as a Photos agent for face recognition and place clustering, or a Finance agent for receipt processing. Each agent is written in Rust using the Spacedrive Developer SDK and then compiled to a secure WASM extension with its own knowledge base and capabilities. This modular approach builds upon research in semantic file systems [**?**], information retrieval [**?**], and ubiquitous computing [**?**].

Extensions provide a secure, sandboxed environment where agents can observe the continuously updated VDFS index, react to events, and propose actions through the same safe, transactional primitives available to human users. This architecture enables a shift from reactive file management to a collaborative model where specialized agents assist with domain-specific tasks while keeping the user in control.

4.6.1 Extension-Based Agents in Practice. Spacedrive's agent architecture is demonstrated through specialized extensions like the Photos extension. When a user enables the Photos extension on their library, an intelligent agent begins operating in the background. As new photos are indexed, the agent receives `EntryCreated` events and adds them to an analysis queue. When the queue reaches a threshold (e.g., 50 photos), the agent dispatches a `AnalyzePhotosBatchJob` that runs face detection, scene classification, and place identification. The agent maintains persistent memory of face clusters, place associations, and scene patterns, enabling sophisticated queries like "show me photos of Alice at the beach."

Each extension agent is sandboxed within a WASM runtime with explicit permissions. The Photos extension, for example, requires permissions to read image files, write sidecars for face detection results, add tags, and use local face detection models. Users grant these permissions on a per-location basis, ensuring the agent only processes photos in directories they've explicitly authorized.

4.6.2 The Agent Architecture: Memory, Events, and Actions. Extension-based agents operate through three core primitives:

Memory Systems. Each agent maintains persistent knowledge across restarts through specialized memory types:

- **Temporal Memory**: Time-ordered event stream supporting temporal queries. For example, `since(timestamp)` retrieves all events after a given time, enabling queries like "photos analyzed in the last week with beach scenes." Events are stored with full timestamps and searchable metadata.
- **Associative Memory**: Semantic knowledge graph supporting similarity-based retrieval. Agents query by semantic similarity with configurable thresholds (e.g., "find faces similar to this cluster with >0.85 similarity"). Enables cross-domain queries like "places where Alice frequently appears"

by traversing relationships between face clusters and GPS locations.

- **Working Memory**: Transactional current state tracking. Maintains active plans, queued operations, and in-progress analysis. Persists across agent restarts, enabling resumable workflows.

These memory systems provide agents with persistent knowledge, transforming them from stateless processors into intelligent assistants that learn and adapt.

Event-Driven Processing. Agents react to the VDFS through declarative lifecycle hooks defined in the SDK:

- `#[on_startup]`: Initialization hook for loading models and restoring state
- `#[on_event(EntryCreated)]`: React to specific filesystem events
- `#[scheduled(cron = "0 2 * * *")]`: Time-based triggers for batch processing
- `#[filter(".extension().is_image()")]`: Entry filtering expressions to target specific file types

This declarative approach enables agents to specify complex reactive behaviors without manual event loop management.

```rust
#[agent]
impl PhotosAgent {
    #[on_startup]
    async fn initialize(&self, ctx: &AgentContext) {
        // Load face detection model and restore memory
    }

    #[on_event(EntryCreated)]
    #[filter(".extension().is_image()")]
    async fn on_new_photo(&self, entry: Entry, ctx: &
 AgentContext) {
        // Queue photo for analysis
        ctx.memory.temporal.record(PhotoDiscovered {
 entry });
    }

    #[scheduled(cron = "0 */4 * * *")]  // Every 4 hours
    async fn batch_analysis(&self, ctx: &AgentContext) {
        // Process queued photos in batches
    }
}
```

Listing 4: Event-driven agent structure using SDK attributes

Safe Actions. Agents propose changes through the same Action System used by human users. When the Photos agent identifies a face cluster, it generates a `BatchTagAction` to tag relevant photos, complete with a preview showing exactly which files will be modified. All agent-initiated actions are auditable, reversible, and require the same validation as user commands. This ensures agents operate within the same safety boundaries as manual operations.

4.6.3 Domain-Specific Intelligence. The extension architecture enables specialized agents tailored to specific domains rather than a single general-purpose assistant. A Finance extension can implement intelligent receipt processing: when a PDF invoice appears in an Ingest location, the Finance agent's `on_event` handler triggers OCR extraction, uses a language model to parse merchant details and amounts, then proposes a `FileMoveAction` to organize it into the appropriate `/Finances/Vendors/[Merchant]` directory structure.

Similarly, a Photos extension implements sophisticated computer vision workflows: face detection with clustering, scene classification, place identification from GPS data, and automatic moment generation. Each domain-specific agent brings specialized models and processing logic while operating within the secure, permission-controlled WASM sandbox.

This modular approach provides several advantages over monolithic AI assistants. Agents can be independently developed, tested, and updated. Users install only the agents relevant to their workflows. Each agent's permissions are explicitly scoped, enhancing security. And domain-specific knowledge enables more accurate and efficient processing than general-purpose models.

4.6.4 Proactive Pattern Recognition. Extension agents leverage their persistent memory systems to identify patterns and suggest optimizations. An Organization agent can observe repeated user actions in the `audit_log`—such as consistently moving screenshots from Desktop to project-specific folders—and proactively offer automation rules. When a new screenshot appears, the agent proposes a `FileCopyAction` with a confidence score based on historical patterns: "I noticed you typically move screenshots like this to `/Projects/DesignWork/Screenshots`. Would you like me to move this one?"

Deduplication Intelligence: A Storage agent can periodically scan for duplicated content across devices and suggest cleanup actions that consolidate files and free space, with the Action System's preview showing exactly which files will be removed and where backups exist.

Data Guardian Monitoring: A Backup agent leverages the Content Identity system (Section 4.2) to monitor file redundancy. When new photos are imported with zero redundancy, the agent generates a suggestion: "I noticed you've added 523 photos that currently exist only on your MacBook. These files could be lost if your device fails. Would you like me to create backups on your NAS and Cloud Storage?" The agent maintains knowledge of backup preferences and device availability in its associative memory, enabling intelligent, context-aware suggestions.

Ingestion Processing. Extension agents are particularly powerful when monitoring Ingest Locations (Section 4.4.9). A Finance agent, for example, can register an `on_event` handler for `EntryCreated` events in designated ingest directories. When a new PDF arrives, the agent:

- Receives the `EntryCreated` event with the new Entry
- Dispatches an OCR job to extract text content
- Analyzes the extracted text to identify the document type (invoice, receipt, statement)
- Queries its associative memory for similar documents and their historical organization patterns
- Proposes a `FileMoveAction` to the appropriate destination directory

The user sees a notification: "New invoice from Acme Corp detected. Move to `/Finances/Invoices/2024/Acme?`" This suggestion is a standard, pre-visualized Action that can be approved with a single click, maintaining human-in-the-loop control while automating repetitive organization tasks.

4.6.5 File Intelligence via Virtual Sidecars. Extension agents enrich the VDFS through the Virtual Sidecar System. Background jobs dispatched by agents use purpose-specific models to extract structured information:

- **Text Embeddings**: Lightweight models like all-MiniLM-L6-v2 for semantic search
- **OCR**: Tesseract or EasyOCR for text extraction
- **Speech-to-Text**: Whisper models for transcription
- **Image Analysis**: CLIP or similar for object/scene detection

These specialized models are far more efficient than general-purpose LLMs while providing superior results for their specific tasks.

Image Object Extraction: An `ImageAnalysisJob` processes image files. Using a multimodal model, it identifies objects and concepts within the image (e.g., "dog," "beach," "sunset"). These results are not stored in a sidecar, but are instead applied directly as Tags to the Entry's `UserMetadata` record. This integrates AI analysis into the user's organizational structure and makes images searchable via existing tag filters.

OCR and Transcription: For images and PDF documents, an `OcrJob` is triggered by relevant agents. It extracts all textual content and saves it to a structured sidecar file (e.g., `ocr.json`). Similarly, a `TranscriptionJob` uses a speech-to-text model on audio and video files to produce a `transcript.json` sidecar. The text content from these sidecars is then ingested into the Temporal-Semantic Search FTS5 index, making the content of non-text files fully searchable. A user can now find a photo of a receipt by searching for the vendor's name, or find a video by searching for a phrase spoken within it.

This system transforms a simple collection of files into a rich, interconnected knowledge base that extension agents can reason about, all while maintaining a local-first, privacy-preserving architecture within secure WASM sandboxes.

4.6.6 Intelligent Storage Tiering. The VDFS's native understanding of `StorageClass` enables Storage extension agents to analyze access patterns and suggest tiering changes. A Storage agent can monitor file access metadata and maintain temporal memory of usage patterns across the library.

Consider Bob, a photographer with a Storage agent installed: The agent's scheduled task runs weekly, querying its temporal memory for files with low access frequency. It identifies RAW photo shoots from 2023, tagged as "delivered," that haven't been opened in six months.

Agent Proposal: "I can re-classify 8 completed photo shoots (1.2TB) as **Cold Storage**, moving them to your NAS archive. This will free up space on your main SSD. These files will remain fully searchable, but access will take longer. Do you approve?"

When Bob approves, the agent dispatches a standard `FileCopyAction` to the job queue. The agent acts as an advisor, but the operation uses the same safe, transparent, and verifiable primitives as manual user commands.

Storage agents can also detect the inverse pattern: when files in `Cold` storage see increased access, the agent can propose promoting them back to `Hot` storage. This human-in-the-loop approach ensures users maintain control while benefiting from intelligent automation.

4.6.7 Privacy and Security Through Sandboxing. The extension-based agent architecture provides strong security guarantees through WASM sandboxing and explicit permission controls. Each extension agent operates in an isolated WASM runtime with access limited to explicitly granted permissions. When a user installs a Photos extension, they must grant specific permissions: read access to image files, write access to face detection sidecars, tag creation privileges, and the right to use local face detection models. These permissions can be further scoped to specific library locations, ensuring the agent only processes files in authorized directories.

All agent processing occurs locally within the WASM sandbox. Face detection models, scene classifiers, and other AI capabilities run on the user's device, ensuring sensitive data never leaves their control. The extension SDK provides access to local model inference through a secure host function interface that enforces rate limits and resource constraints.

This architecture separates concerns between lightweight embedding models for search (which operate on the core VDFS) and specialized agent intelligence (which runs in isolated extensions). The AI provider interface supports multiple deployment models: local processing via Ollama for complete privacy, cloud-based services for enhanced capabilities, and enterprise self-hosted solutions for organizational control.

This approach creates a personal, private data space where AI capabilities enhance functionality without compromising control, privacy, or security.

Ethical Considerations. While model-agnostic, Spacedrive prioritizes ethical AI use. Local models mitigate bias by training on user data only, but users are notified of potential limitations (e.g., underrepresented demographics in embeddings). Cloud options include opt-out for sensitive files, ensuring compliance with regulations like GDPR.

4.7 Temporal-Semantic Search: An Asynchronous, Job-Based Architecture

Key Takeaways

- **Asynchronous by Design**: All searches are durable jobs, preventing UI blocking and enabling extension agents to initiate searches without waiting for results.
- **Hybrid Architecture**: A high-speed FTS5 keyword filter pre-processes candidates before lightweight semantic re-ranking.
- **Decoupled Results**: The backend caches results against a query ID; the frontend is notified to fetch and render them, decoupling agents from the UI.

Spacedrive's search architecture is engineered for a responsive, agent-driven environment. It treats search not as a synchronous request-response loop, but as a durable, asynchronous job that is initiated, executed, and cached by the backend, with the frontend being notified upon completion. This model applies uniformly to searches originating from both direct user input and extension agent directives.

4.7.1 *The Search Lifecycle: An Action-Driven Workflow.*

A search operation follows a formal lifecycle managed by core VDFS components:

(1) **Action Dispatch**: A user or extension agent initiates a search by dispatching a `SearchAction` containing the query parameters.
(2) **Job Registration**: The `ActionManager` receives this action and registers a new `SearchJob` with the Durable Job System, returning a unique `query_id`. This step is near-instantaneous, providing immediate feedback.
(3) **Asynchronous Execution**: The `SearchJob` executes in the background, performing the two-stage search process without blocking the user or agent.
(4) **Result Caching**: Upon completion, the job caches the ordered list of resulting `Entry` IDs in a temporary store, keyed by the `query_id`.
(5) **Frontend Notification**: The Job System emits a `SearchResultsReady(query_id)` event to the system's event bus.
(6) **Result Rendering**: The frontend, subscribed to these events, receives the notification and uses the `query_id` to fetch the cached results via the internal API for rendering.

This architecture ensures an agent's role is simply to initiate the search; it never needs to handle the results directly, maintaining a clean separation of concerns.

4.7.2 *The Two-Stage Search Process.*

The `SearchJob` executes a hybrid temporal-semantic query designed for performance on consumer hardware. This **Temporal-First, Vector-Enhanced** approach operates in two stages:

(1) **Hierarchical Pre-filtering (Closure Table)**: If the search query includes a path scope (e.g., "find photos in 'Projects'"), the system first uses the closure table to get a list of all entry IDs within that scope. This provides a highly-focused candidate set before any text search occurs.
(2) **Temporal Filtering (FTS5)**: Next, SQLite's FTS5 index performs a high-speed keyword search only on the pre-filtered candidate set. This rapidly narrows millions of entries down to a small, relevant set, typically in under 55ms.
(3) **Semantic Re-ranking**: For queries requiring semantic understanding, the system computes a vector embedding of the query using a lightweight local model (e.g., `all-MiniLM-L6-v2`). It then re-ranks only the candidate set from Stage 2 by cosine similarity against their pre-computed embeddings, adding minimal latency (typically <40ms).

This hybrid process provides the power of semantic search with the speed of traditional indexed search, delivering sub-100ms response times on consumer hardware for libraries with over a million entries.

4.7.3 *Unified Vector Repositories: Distributed Semantic Intelligence.*

Unlike traditional vector databases that centralize embeddings in a monolithic index, Spacedrive employs a distributed system of Unified Vector Repositories—adaptive sidecars that combine routing intelligence with content embeddings to enable efficient semantic search at scale.

Unified Vector Repository Architecture. Instead of separate routing nodes and content stores, Spacedrive uses a single, standardized Vector Repository format. These repositories are created adaptively throughout the filesystem and can contain multiple collections, allowing them to serve different roles within a single, portable format.

Efficient Embedding Models. Crucially, Spacedrive employs lightweight embedding models—not large language models—for semantic search:

- **all-MiniLM-L6-v2**: 22M parameters, 384-dimensional vectors, 5MB model size
- **nomic-embed-text-v1.5**: 137M parameters, optimized for retrieval tasks
- **BGE-small-en**: 33M parameters, excellent performance/size ratio

These models run efficiently on CPU, produce embeddings in milliseconds, and require minimal memory—making real-time semantic indexing practical during file discovery. The models are small enough to bundle with Spacedrive (under 100MB total) and fast enough to process thousands of files per second on consumer hardware.

```json
{
  "version": 3,
  "path": "/Projects/spacedrive-core",
  "collections": {
    "routing": {
      "keywords": ["rust", "filesystem", "vdfs", "sqlite", "p2p"],
      "child_hints": {
        "/src": ["implementation", "core", "indexer", "networking"],
        "/docs": ["documentation", "architecture", "rfc"],
        "/tests": ["testing", "integration", "unit", "benchmarks"]
      },
      "aggregate_embedding": [0.123, -0.456, ...],
      "descendant_count": 3847
    },
    "content": {
      "file_embeddings": {
        "README.md": {
          "model": "nomic-embed-text-v1.5",
          "vector": [0.234, 0.567, ...],
          "summary": "Main project documentation and setup guide"
        },
        "Cargo.toml": {
          "model": "all-MiniLM-L6-v2",
          "vector": [0.345, 0.678, ...],
          "keywords": ["dependencies", "workspace", "edition"]
        }
      }
    },
    "metadata": {
      "density_score": 0.85,
      "last_updated": "2024-03-15T10:30:00Z",
      "access_frequency": 234
    }
  }
}
```

Listing 5: Example Vector Repository structure

Adaptive Repository Creation. Vector Repositories are not created for every folder. The system places them based on semantic density and usage patterns:

- **Location Roots**: Created at the root of each Location as primary entry points
- **Semantic Density**: Folders reaching thresholds of file count and semantic richness
- **Content Divergence**: When child folders contain semantically distinct content
- **Project Boundaries**: Auto-detected via markers (package.json, .git, Cargo.toml)
- **User Patterns**: Created for frequently searched or bookmarked folders

Lightweight Routing Without LLMs. The routing collection uses statistical methods and lightweight embeddings—not large language models—for efficient navigation:

(1) **TF-IDF Keyword Extraction**: Statistical term frequency analysis identifies distinctive terms per folder
(2) **Child Hints**: Simple keyword lists derived from filenames and content sampling
(3) **Aggregate Embeddings**: Computed using efficient models like all-MiniLM-L6-v2 (22M params, runs on CPU)
(4) **Progressive Traversal**: Cosine similarity scores guide path selection without neural network inference

Search Flow Example. When searching for "rust async file watcher implementation":

(1) Root repository's routing collection identifies /Projects as 85% relevant
(2) /Projects repository routes to /spacedrive-core (92% match on "rust", "filesystem")
(3) /spacedrive-core repository suggests /src (keywords: "implementation", "core")
(4) /src repository pinpoints /location/watcher subdirectory
(5) Content embeddings in final repository provide exact file matches
(6) Meanwhile, lower-scoring paths (70% matches) continue processing in background

Performance and Scalability Benefits. This architecture provides several key advantages:

- **Memory Efficiency**: Load only relevant repositories, not entire vector database
- **Incremental Updates**: Only affected repositories need recomputation
- **Natural Sharding**: Filesystem hierarchy provides logical partitioning
- **Offline Capability**: Each device has complete semantic search of local content
- **Progressive Enhancement**: Repositories evolve from simple to more capable as needed

The unified format ensures all intelligence—routing and content vectors—travels with the data, while the adaptive creation strategy prevents overhead in sparse areas of the filesystem. This enables million-file semantic search on consumer hardware by transforming an O(n) problem into an O(log n) traversal guided by semantic routing.

Integration with Extension Agents. The Vector Repository system integrates with Spacedrive's extension agents, enabling them to:

- Navigate large filesystems using routing hints
- Understand folder purposes through aggregate embeddings
- Provide natural language summaries of search results by traversing the semantic hierarchy
- Learn optimal repository placement from user search patterns

This distributed approach represents a fundamental innovation in semantic search architecture, making AI-powered file discovery practical at any scale while maintaining the portability and privacy benefits of Spacedrive's local-first design.

4.8 Native Storage Tiering: Reconciling Physical Reality and User Intent

Key Takeaways

- **Hybrid Model**: Distinguishes between a Volume's physical capabilities and a Location's logical purpose
- **Smart Classification**: Automatically identifies and filters user-relevant storage volumes
- **Warning System**: Prevents performance surprises by reconciling user intent with physical limitations

Spacedrive's VDFS integrates an understanding of storage tiers that distinguishes between a **Volume's physical capabilities** and a **Location's logical purpose**. This allows the system to honor user intent while grounding operations in physical reality, preventing performance bottlenecks and providing actionable warnings.

This is achieved through a dual-property system:

- **PhysicalClass (on the Volume)**: An automatically detected property that reflects the hardware's nature. The Volume Classification System infers this by benchmarking performance (e.g., SSDs are classified as `Hot`) or querying cloud provider APIs (e.g., an AWS Glacier bucket is classified as `Cold`).
- **LogicalClass (on the Location)**: A user-defined property that reflects organizational intent. A user can mark any Location as `Hot`, `Warm`, or `Cold` to signify its purpose (e.g., a "/Projects/Archive" folder on a fast SSD can be marked as `Cold`).

```rust
pub enum StorageClass {
    Hot,   // e.g., Local SSD, standard cloud storage
    Warm,  // e.g., S3 Infrequent Access
    Cold,  // e.g., AWS Glacier Instant Retrieval
    Deep,  // e.g., Glacier Deep Archive
}

pub struct Volume {
    // ... existing fields ...
    pub physical_class: StorageClass,
}

pub struct Location {
    // ... existing fields ...
    pub logical_class: Option<StorageClass>,
}

// Determine effective storage class for operations
impl Location {
    pub fn effective_storage_class(&self) -> StorageClass
      {
        match (self.volume.physical_class, self.
    logical_class) {
            (physical, Some(logical)) => {
                // Take the more restrictive (colder)
    class
                std::cmp::max(physical, logical)
            },
            (physical, None) => physical,
        }
    }
}
```

Listing 6: Hybrid storage classification model

4.8.1 *Determining the Effective StorageClass.*

For any given file, the VDFS determines its **Effective StorageClass** by taking the more restrictive (colder) of the two properties. This ensures physical limitations are respected.

4.8.2 *Impact on Core Systems.*

The Effective `StorageClass` informs the behavior of the entire VDFS:

- **Action System**: The simulation engine uses this to generate warnings when there is a mismatch between logical and physical classes, preventing user frustration (Section 4.4).
- **SdPath Resolver**: When resolving a content-addressed path, the system prioritizes sources with a `Hot` Effective StorageClass to ensure optimal performance.
- **Data Guardian**: The AI can enforce redundancy policies based on the Effective StorageClass, such as requiring at least one `Hot` copy of critical files before allowing any to be moved to `Cold` storage.

Volume Characteristics

Spacedrive automatically discovers and tracks key properties of each storage device:

- **Hardware Type**: SSD vs. HDD vs. Network storage for optimization decisions
- **Performance Metrics**: Measured read/write speeds to inform file operations
- **Role Classification**: Primary drive, external storage, or system volume
- **Advanced Features**: Copy-on-write filesystem support for instant large file operations

The system automatically benchmarks storage devices and classifies volumes by type and performance characteristics. Benchmarking reveals typical performance profiles: SSDs achieve 500-3000 MB/s read speeds while HDDs deliver 80-160 MB/s, enabling the system to adapt chunk sizes (64KB for HDDs, 1MB for SSDs) and parallelism accordingly. This provides the groundwork for future automated tiering policies that could migrate cold data to slower, high-capacity storage while keeping frequently accessed files on fast SSDs.

4.9 Volume Classification

Spacedrive employs a **Volume Classification System** that provides platform-aware storage management, improving user experience while reducing system overhead:

4.9.1 *Platform-Aware Volume Types.*

Rather than treating all storage as equivalent, Spacedrive classifies volumes based on their actual role and user relevance:

The system employs a volume type taxonomy (Primary, UserData, External, Secondary, System, Network, Unknown) with platform-specific classification logic. For example, macOS classification recognizes the root filesystem, dedicated user data volumes, system-internal volumes, and external mounts based on mount point patterns, enabling filtering of user-relevant storage.

4.9.2 *Auto-Tracking.*

The classification system enables **smart auto-tracking** that focuses on user-relevant storage:

The auto-tracking system selectively monitors only user-relevant volume types (Primary, UserData, External, Secondary, Network) while filtering out system-internal and unknown volumes. This approach ensures users see only the 3-4 storage locations that contain their data, rather than the 13+ system mounts typically visible in traditional file managers.

User experience improvements: - **Reduced visual clutter**: Users see 3-4 relevant volumes instead of 13+ system mounts - **Automatic relevance filtering**: System volumes (VM, Preboot, Update partitions) hidden by default - **Cross-platform consistency**: Unified volume semantics across macOS APFS containers, Windows drive letters, and Linux mount hierarchies - **Performance optimization**: Eliminates unnecessary indexing of system-only volumes

4.9.3 *Platform-Specific Optimizations.*

The system handles complex platform-specific storage architectures using rules and heuristics:

macOS APFS Containers: Recognizes that `/System/Volumes/Data` contains user files even though `/` is the system root, properly classifying the sealed system volume separately from user data.

Windows Drive Management: Distinguishes between primary system drives (C:), secondary storage (D:, E:), and hidden recovery partitions, presenting a clean drive letter interface to users.

Linux Mount Complexity: Filters virtual filesystems (`/proc`, `/sys`, `/dev`) and container mounts while properly identifying user-relevant storage like `/home` partitions and network mounts.

This platform-aware approach transforms the overwhelming technical complexity of modern storage systems into an intuitive, user-friendly interface that focuses attention on storage that actually contains user data.

4.10 Platform Integrations: Unified Access to Traditional and Modern Storage

Spacedrive's VDFS extends beyond local filesystems to natively integrate traditional protocols like FTP, SMB, and WebDAV, as well as modern cloud storage. Third-party cloud storage services (S3, Google Drive, Dropbox, etc.) are implemented as first-class **volumes** within the VDFS, distinct from device-local locations. This architectural decision reflects the fundamental difference between cloud services (which are API-accessed storage backends) and local filesystems.

The system leverages OpenDAL to provide unified access to over 40 cloud providers. Each cloud service appears as a volume with its own dedicated `SdPath::Cloud` addressing variant, enabling the full power of the Spacedrive Indexing Engine—including adaptive hashing via ranged reads for efficient content identification—without downloading entire files. This is separate from the Spacedrive Cloud

Volume PhysicalClass	Location LogicalClass	Effective StorageClass	System Behavior & User Feedback
`Hot` (SSD)	`Hot`	`Hot`	Normal, high-performance operation
`Hot` (SSD)	`Cold`	`Cold`	User's archival intent is respected
`Cold` (HDD)	`Cold`	`Cold`	Normal archival operation
`Cold` (HDD)	`Hot`	`Cold`	**Physical limits override user intent**. Warning issued during preview

Table 5: Effective StorageClass determination and system behavior

Core concept (Section 5), which refers to hosting Spacedrive itself as a managed service.

This cloud-as-volume architecture ensures:

- Proper path handling for cloud-native formats (e.g., S3's bucket/key structure)
- Consistent content hashing across all storage types for true cross-storage deduplication
- Transparent operations between local, cloud, and cross-cloud transfers

5 Architectural Application: A Native Cloud Service

The flexibility of the Spacedrive V2 architecture is best demonstrated by its application in creating a cloud service that natively integrates with the user's personal P2P network. Unlike traditional cloud backends that require custom APIs and treat the server as a privileged entity, our model treats the cloud instance as just another Spacedrive device. This approach leverages the core VDFS abstractions to provide cloud storage that feels native, secure, and seamlessly integrated into the user's existing ecosystem.

5.1 Core Principle: Managed Cores as First-Class Devices

The foundational principle of the Spacedrive Cloud Service is that each user is provisioned a managed, containerized instance of the unmodified `sd-core` engine. This managed instance—which we refer to as a "Cloud Core" for convenience—is architecturally identical to any other Spacedrive core. It has its own unique device ID, participates in the same P2P network as the user's other devices, and exposes its storage as standard Spacedrive Locations. The term "Cloud Core" simply denotes its deployment context (managed hosting), not any special software or capabilities.

This design offers profound architectural advantages:

- **Zero Custom APIs**: All interactions with the Cloud Core, from file transfers to metadata sync, use the exact same Iroh-powered protocols as any other peer-to-peer connection. There is no separate "cloud API".
- **Native Device Semantics**: The Cloud Core is cryptographically a standard device. It must be paired and trusted just like a user's phone or laptop, inheriting the entire security and trust model of the core architecture.
- **Location Abstraction**: Cloud storage is not a special case. It is simply a Location (e.g., `/cloud-files`, `/backups`) within the Cloud Core's VDFS, making it universally addressable via `SdPath`.

5.2 Integration and User Experience

From the user's perspective, integrating cloud storage is indistinguishable from adding a new physical device. The connection flow leverages the same native pairing process: a user's local Spacedrive client initiates pairing, and the newly provisioned Cloud Core joins the session using the provided code.

Once paired, the Cloud Core appears in the user's device list alongside their other machines. Operations that span local and cloud storage become trivial. For example, copying a local file to the cloud is a standard `FileCopyAction` where the destination `SdPath` simply references the Cloud Core's device ID:

```rust
// Copy a local document to the "Cloud Files" location
// on the user's provisioned cloud device.
copy_files(
    vec![SdPath::local("~/Documents/report.pdf")],
    SdPath::new(cloud_device_id, "/data/cloud-files/")
).await?;
```

Listing 7: A cross-device copy to the cloud uses the same native operation

This demonstrates the power of the VDFS abstraction. The underlying complexity of the network transfer is handled by the unified networking layer and the durable job system, making the cloud a natural extension of the user's personal data space.

This integration extends to fundamental operations like file uploads. A user accessing their Library via the Spacedrive web application can upload files directly from their browser. This action does not require a custom API endpoint; instead, it triggers the native **Ingestion Workflow** (Section 4.4.9). The browser initiates a secure transfer using the same Iroh-powered P2P protocols, sending the file directly to the user's designated Ingest Location on a preferred device—which could be their Cloud Core instance itself. This illustrates the benefits of the unified architecture: a simple drag-and-drop in a web browser translates into a durable, transactional operation within the user's private VDFS, managed by the core engine.

5.3 Cloud-Native Architecture and Data Isolation

The service is designed to be Kubernetes-native, leveraging container orchestration for scalability, resilience, and security. Each Cloud Core runs in its own isolated Pod, ensuring strict user data separation.

User data persistence is managed through per-user Persistent Volume Claims (PVCs), which map to encrypted cloud block storage (e.g., AWS EBS, Google Persistent Disk). This architecture ensures that a user's entire cloud instance—their library database, storage locations, and configuration—is a self-contained and portable unit.

Kubernetes NetworkPolicies are employed to enforce cryptographic isolation at the network level. Each user's pod is firewalled to only allow traffic from other devices within their trusted P2P network, effectively extending the private network into the cloud environment.

5.4 Benefits of the Hybrid Model

This architectural approach provides the benefits of both local-first and cloud-based systems:

- **High Availability**: The Cloud Core can act as an online peer, enabling asynchronous operations like backups or file sharing even when local devices are offline.
- **Centralized Backup Target**: Users can configure local Locations to automatically back up to a Location on their Cloud Core.
- **Asynchronous Sharing**: The cloud instance can act as a relay for Spacedrop transfers, where a user uploads a file once to get a shareable link through the `sd.app` domain (or custom domains for self-hosted instances).

5.5 Enterprise Deployment and Data Sovereignty

The same architectural principles that enable the native cloud service provide a direct path for on-premise enterprise deployments. The containerized, Kubernetes-native design of the "Cloud Core" allows organizations to deploy Spacedrive entirely within their own infrastructure, achieving complete data sovereignty while maintaining the user-friendly experience.

5.5.1 *On-Premise Architecture.*

In an enterprise deployment, organizations run their own Spacedrive backend infrastructure:

- **Identity Integration**: Native support for LDAP, Active Directory, and OAuth2/SAML providers
- **Storage Integration**: Integration with existing enterprise storage (SAN, NAS, S3-compatible object stores)
- **Deployment Flexibility**: Support for bare metal, VMware, OpenStack, or Kubernetes environments
- **Geographic Distribution**: Multi-site deployments with policy-based routing between locations

5.5.2 *Team Libraries and Collaboration.*

The architecture naturally extends to support collaborative workflows:

- **Shared Libraries**: Teams can create collaborative Libraries with fine-grained access control
- **Role-Based Access Control**: Full-featured RBAC system built on the Action System foundation
- **Department Isolation**: Cryptographic separation between different organizational units
- **Audit Trail**: Every action logged with full attribution for compliance and security

5.5.3 *Enterprise Features.*

Additional capabilities designed for organizational needs:

- **Compliance Controls**: Data retention policies, legal hold, and audit log exports
- **Advanced Analytics**: Usage patterns, storage optimization recommendations, and cost allocation
- **API Access**: A JSON-RPC based API for integration with existing enterprise tools and a full-featured Command Line Interface (CLI)
- **Professional Support**: SLA-backed support with dedicated account management

This enterprise model demonstrates how Spacedrive's core architecture—designed for individual user empowerment—scales naturally to organizational deployment without compromising its fundamental principles of user control, data sovereignty, and intuitive operation.

5.6 Collaboration and Public Sharing

The VDFS architecture enables sharing capabilities by leveraging any core instance configured for public access. A core can act as an available peer, a public file host, or a transfer relay, depending on its deployment and configuration. The Spacedrive Cloud service provides a managed, pre-configured core for these roles, but any self-hosted core can be set up to perform the same functions.

5.6.1 *Flexible Hosting Model.*

While Spacedrive Cloud provides turnkey hosting, the architecture supports multiple deployment options:

- **Spacedrive Cloud**: Managed hosting with automatic SSL, CDN, and scaling
- **Self-Hosted Core**: Deploy on any infrastructure with full control
- **Hybrid Deployment**: Mix of self-hosted and managed components
- **Edge Deployment**: Run cores close to users for optimal performance

Any Spacedrive core—whether on a personal device or in the cloud—can serve as a sharing endpoint with appropriate configuration.

5.6.2 Shared Folders via Team Libraries. Collaboration in Spacedrive leverages the Library abstraction:

- **Team Libraries**: Shared libraries with role-based permissions
- **Granular Access Control**: Per-location and per-file permissions
- **Action Audit Trail**: Complete history of all modifications
- **Conflict Resolution**: Automatic handling of concurrent edits

Team members connect to shared libraries exactly as they would personal ones—an always-available peer, such as a managed Cloud Core or a user's own self-hosted server, ensures data availability for the team.

5.6.3 Public File Hosting. Spacedrive's architecture allows any core to serve files publicly. The distinction in user experience comes from the network configuration:

- **Personal Device Core**: A user can serve files directly from their laptop or NAS, but this requires manual setup like port forwarding on their router and configuring Dynamic DNS to handle changing IP addresses.
- **Self-Hosted Core**: An organization can deploy a core on their own server with a static IP and manage their own SSL certificates and web server configuration.
- **Spacedrive Cloud (Managed Core)**: For convenience, the Spacedrive Cloud service automates this. It runs the user's core behind a managed load balancer that handles SSL termination and provides a stable public URL (e.g., `https://sd.app/user/file.pdf`). This is a feature of the managed service, not a special capability of the core software itself.

```
# Via Spacedrive Cloud (automatic SSL + CDN)
https://sd.app/user/file.pdf

# Via self-hosted server core
https://files.company.com/public/
    presentation.pdf

# Via personal device (requires port forwarding)
https://home.user.com:8443/share/
    document.docx
```

Listing 8: Public sharing URL examples

5.6.4 Enhanced Spacedrop. A Spacedrive core configured as a **public relay** can extend Spacedrop's capabilities. While a standard P2P transfer is ephemeral, using a relay enables:

- **Asynchronous Transfers**: The relay can hold files until recipients connect, meaning the sender can go offline after uploading.
- **Persistent Links**: Links to the relay remain valid, turning an ephemeral transfer into a shareable resource.
- **Large File Support**: No size limits with resumable transfers
- **Access Control**: Optional passwords and expiration dates

Users can deploy their own relays or use the one provided by the Spacedrive Cloud service (e.g., `https://drop.spacedrive.com/...`).

```
# Direct P2P (ephemeral, no relay)
spacedrop://device-id/transfer-id

# Via Spacedrive Cloud relay
https://drop.spacedrive.com/abc123

# Via self-hosted relay
https://relay.company.com/drop/xyz789
```

Listing 9: Spacedrop relay options

This unified approach to sharing—from private team collaboration to public content distribution—demonstrates how core P2P primitives scale to support diverse use cases without architectural compromises.

6 Implementation and Evaluation

Key Takeaways

- **Production-Ready**: Built in Rust with memory safety guarantees • 95%+ test coverage • Multi-process distributed testing framework
- **Compatibility**: Works with existing filesystems, cloud services and tools.

Spacedrive's design principles are validated through careful implementation choices and rigorous performance analysis.

6.1 Technology Stack

Spacedrive is implemented in **Rust** to leverage its guarantees of memory safety, performance, and fearless concurrency—essential for a reliable, multi-threaded distributed system. The core technology stack reflects modern best practices:

- **Tokio Runtime**: Async execution with work-stealing scheduler for efficient I/O handling
- **SeaORM with SQLite**: Type-safe database operations with ACID transactions and FTS
- **O(1) Hierarchical Lookups**: Directory listings, subtree calculations, and ancestor lookups are performed in constant time via indexed closure table joins, replacing slow LIKE queries and scaling to millions of entries
- **Event-Driven Architecture**: Custom EventBus for loose coupling and state propagation
- **Job System**: MessagePack-serialized tasks with automatic resumability
- **Daemon Architecture**: Flexible deployment via Unix domain sockets with JSON-RPC protocol, supporting both embedded and daemon modes
- **FFmpeg Integration**: Bundled media processing libraries for thumbnail generation and metadata extraction
- **OpenDAL Integration**: Unified storage layer supporting 40+ cloud and object storage services with consistent API

.1 Database Schema Optimization

The database design prioritizes both space efficiency and query performance through several key optimizations:

Table 6: Performance benchmarks across storage tiers. Testing conducted on M2 MacBook Pro with 16GB RAM running macOS 14.5. Values generated from the benchmarking harness summaries.

	Small Dataset		Medium Dataset		Large Dataset	
Phase	**HDD**	**SSD**	**HDD**	**SSD**	**HDD**	**SSD**
Discovery	2.7	1.4	22.7	7.4	58.7	60.2
Processing	1.2	1.2	6.1	6.1	60.0	52.9
Content Identification	38.5	68.7	191.6	265.8	2922.7	1340.8

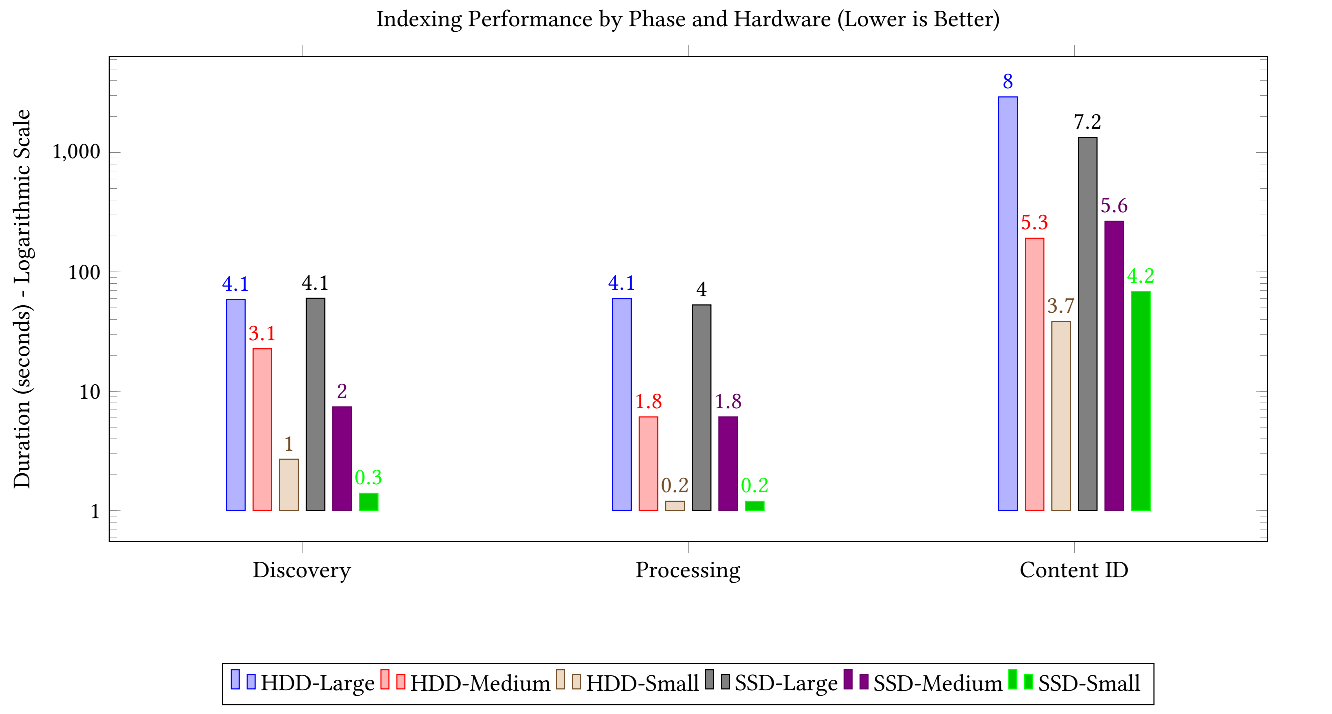


Figure 6: Comparison of indexing duration across different phases, hardware types, and dataset sizes. Note the logarithmic scale for the Y-axis to accommodate the wide range of timings. The most significant performance difference is in the I/O-bound Content Identification phase, where the SSD significantly outperforms the HDD on the large dataset.

.2 Testing and Validation Framework

Spacedrive employs a thorough testing strategy designed for real-world scenarios:

.2.1 *Multi-Process Test Framework.*

A custom Cargo-based subprocess framework enables testing of distributed scenarios:

Multi-Process Distributed Testing

Spacedrive employs a testing framework that simulates real-world distributed scenarios:

Role-Based Testing

- Tests run multiple processes simultaneously, each taking a different device role (Alice, Bob, etc.)
- Environment variables control which role each process assumes during testing
- Enables authentic multi-device interaction testing without requiring physical hardware

Realistic Scenario Simulation

- Device pairing processes tested across different network conditions
- File synchronization verified with actual data transfer and conflict resolution
- Network interruption and recovery scenarios validated automatically

Extensive Coverage

- Complex multi-device pairing scenarios with authentication verification
- Cross-platform compatibility testing (macOS, Windows, Linux, mobile)
- Performance validation under various load conditions

Note: Cloud storage indexing uses metadata-only requests with on-demand content fetching. Performance varies based on API rate limits and network conditions.

These benchmarks validate that Spacedrive maintains sub-100ms response times for typical user operations even with multi-million entry libraries, achieving performance previously limited to enterprise systems.

.2.2 Resource Impact Analysis. The performance overhead of Spacedrive's advanced features is carefully optimized:

- **CPU Usage**: Background indexing uses <5% CPU on modern processors, with adaptive throttling on battery
- **Storage Cost**: 250MB database for 1M files represents <0.1% overhead on typical 256GB+ drives
- **Network Efficiency**: P2P transfers eliminate redundant cloud uploads, saving 50% bandwidth for multi-device users
- **Battery Impact**: Mobile devices see <2% additional battery drain with scheduled tasks

These metrics indicate that Spacedrive provides enterprise-oriented capabilities while maintaining consumer-friendly resource usage.

.3 Compatibility and Interoperability

Spacedrive is designed as a layer atop existing storage systems, not a replacement. This philosophy ensures integration with users' current workflows while providing enhanced capabilities.

.3.1 Traditional Filesystem Integration. Spacedrive maintains full compatibility with native filesystems through several key design decisions:

- **Non-invasive indexing**: Files remain in their original locations with native filesystem attributes intact. Spacedrive never modifies file content or filesystem-level metadata during indexing.
- **Filesystem-aware operations**: The system respects platform-specific constraints (e.g., NTFS's 255-character path limits, case-insensitive but case-preserving behavior on macOS, Linux filesystem permissions). Volume detection adapts to each platform's conventions.
- **Transparent file access**: Applications continue accessing files through standard OS APIs. Spacedrive acts as an index and orchestrator, not a filesystem driver or FUSE layer.
- **Preserved compatibility**: Special files (symlinks, junction points, device files) are cataloged but not followed during indexing, preventing circular references while maintaining awareness of filesystem structure.

.3.2 Cloud Service Integration. Spacedrive integrates with cloud services through direct remote indexing rather than traditional sync-based approaches. This enables management of petabyte-scale cloud libraries on devices with minimal local storage by treating cloud storage as standard VDFS Locations. The technical implementation leverages OpenDAL for unified access across providers and employs the same indexing pipeline detailed in Section 4.3.6, ensuring consistent functionality whether data resides locally or in the cloud.

.3.3 Ecosystem Tool Compatibility. Spacedrive enhances rather than replaces existing tools:

- **Standard protocols**: While Spacedrive doesn't expose a traditional mount point, it provides export capabilities to generate file lists compatible with tools like `rsync`, `rclone`, or backup software.
- **Metadata preservation**: Extended attributes (xattrs), alternate data streams (ADS on NTFS), and resource forks (macOS) are preserved during Spacedrive operations, ensuring compatibility with specialized applications.
- **Integration APIs**: A JSON-RPC API and CLI enable automation tools to query the Spacedrive index, trigger jobs, and monitor operations programmatically.
- **Export formats**: Search results and file lists can be exported in common formats (CSV, JSON, file paths) for processing by external tools or scripts.

This interoperability approach ensures Spacedrive complements users' existing toolchains while providing a unified view and management layer across all storage locations.

.4 Scalability Limits and Architectural Boundaries

While Spacedrive is designed for impressive scale, understanding its limits helps in deployment planning:

.4.1 Theoretical Scaling Limits. Based on architectural analysis and stress testing:

Library Size Limits:
- **Maximum Entries**: 100M+ files per library (SQLite page limit)
- **Maximum Devices**: 1,000 paired devices per library
- **Maximum Locations**: 10,000 locations across all devices
- **Database Size**: Up to 1TB with current schema (4KB page size)

Performance Degradation Curves:
- Linear search performance up to 10M entries
- Sub-linear degradation from 10M-50M entries
- Noticeable lag beyond 50M entries without sharding
- Memory usage scales at 150 bytes per entry

.4.2 Practical Deployment Limits. Real-world limits based on hardware constraints:

Consumer Hardware (8GB RAM):
- Comfortable: 1-5M files
- Functional: 5-20M files
- Constrained: 20M+ files

Professional Hardware (32GB+ RAM):
- Comfortable: 10-50M files
- Functional: 50-100M files
- Requires optimization: 100M+ files

.4.3 Strategies for Extreme Scale. For deployments exceeding these limits:

Database Sharding:
- Horizontal partitioning by device or location
- Federated queries across shards
- Consistent hashing for shard distribution

Tiered Architecture:
- Hot data in primary database
- Cold data in archive databases
- Transparent query routing

.5 Failure Recovery Scenarios

Spacedrive's architecture includes recovery mechanisms:

.5.1 Database Corruption Recovery. When database corruption is detected:

Automatic Recovery Process:

(1) Detection via SQLite integrity check on startup
(2) Attempt automatic repair using SQLite recovery tools
(3) If repair fails, restore from automatic backups
(4) Re-index affected locations to ensure consistency
(5) Sync with other devices to restore missing metadata

Manual Recovery Options:

- Export readable data to new library
- Selective location re-indexing
- Point-in-time recovery from backups
- Cross-device metadata reconstruction

.5.2 Partial Sync Failure Handling. When synchronization is interrupted:

Automatic Resume:

- Sync state persisted every 1000 operations
- Automatic retry with exponential backoff
- Conflict detection and resolution on resume
- Progress notification to user

Manual Intervention:

- Force full resync option
- Selective sync for specific domains
- Conflict resolution UI for complex cases
- Sync history for debugging

.5.3 Network Partition Recovery. When devices are separated by network issues:

Partition Detection:

- Heartbeat timeout (30 seconds)
- Quorum detection for device groups
- Automatic operation queuing

Healing Process:

- Vector clock comparison on reconnection
- Automatic merge of non-conflicting changes
- User notification for conflicts
- Full audit trail of partition period

.5.4 Catastrophic Failure Recovery. For complete system failures:

Library Reconstruction:

(1) Create new library from backup
(2) Re-pair devices using recovery keys
(3) Re-index all locations
(4) Restore user metadata from sync
(5) Verify content integrity via hashes

Data Verification:

- Compare content hashes across devices
- Identify missing or corrupted files
- Generate recovery report
- Automated repair where possible

.6 Extensibility Architecture

Spacedrive employs an extensibility model where powerful extensions are written in Rust using a declarative, attribute-based Software Development Kit (SDK). These extensions are then compiled to WebAssembly (WASM) to run in a secure, sandboxed environment, providing both power and safety. This architecture allows extensions to feel like first-class modules of the core application, with deep integration into the data model, sync system, and user interface.

.6.1 WebAssembly Plugin System. For lightweight extensions and custom functionality, Spacedrive employs a WebAssembly-based plugin system:

Security Model. WASM provides critical security guarantees:

- **Complete Sandboxing**: Plugins cannot access filesystem or network without permission
- **Capability-Based**: Plugins declare required permissions upfront
- **Resource Limits**: CPU, memory, and I/O are bounded
- **Memory Safety**: Prevents buffer overflows and pointer manipulation

Plugin Capabilities. Through the Rust SDK, extensions can:

- Define custom data models that become real, indexed SQL tables at runtime
- Implement long-running, durable background jobs and tasks using `#[job]` and `#[task]` attributes
- Create autonomous AI agents that can observe events and dispatch actions
- Integrate directly with Spacedrive's native sync system, with configurable strategies per model
- Add new elements to the user interface, such as sidebar sections and custom views, via a declarative `ui_manifest.json` file

```rust
// Define a 'Person' model for a simple CRM extension
#[model(
    table_name = "person",
    sync_strategy = "shared" // Automatically syncs
     across devices
)]
pub struct Person {
    #[primary_key]
    pub id: Uuid,
    pub name: String,
    #[indexed]
    pub email: Option<String>,
    // Links to core metadata system, enabling tags and
     collections
    #[metadata]
    pub metadata_id: i32,
}
```

Listing 10: Example extension model definition using the Rust SDK

Distribution Model. The WASM approach solves critical distribution challenges:

- **Single Binary**: One .wasm file works on all platforms
- **No Code Signing**: Avoids platform-specific signing requirements
- **Instant Loading**: No process spawn overhead

- **Hot Reload**: Plugins can be updated without restart

.6.2 Plugin Architecture. The WASM plugin system provides flexible extensibility:

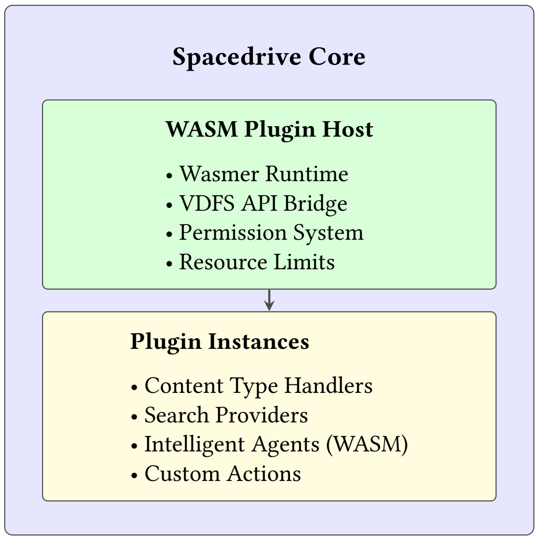


Figure 7: Spacedrive Core WASM Plugin Architecture: Sandboxed execution environment with unified API for all extension types.

This architecture provides:

- Complete sandboxing for all extensions
- Unified API for all plugin types
- Hot-reload capability for development
- Platform-independent distribution

Cloud Storage Integration. Cloud storage providers (S3, Google Drive, Dropbox, etc.) are implemented as WASM plugins that leverage the OpenDAL library. This approach:

- Maintains security through WASM sandboxing
- Enables hot-swappable cloud provider support
- Allows community-contributed storage backends
- Provides consistent API across all storage types

.7 Extensibility Beyond Storage: Data Ingestion Agents

While the VDFS architecture excels at unifying existing filesystems and storage protocols, its extensibility model enables a far more profound capability: the ingestion of disparate data from third-party services through specialized **Data Ingestion Agents**. This transforms Spacedrive from a system that merely organizes a user's files into a comprehensive, private warehouse for their entire digital life.

These agents are lightweight, sandboxed WASM plugins that act as connectors to external data sources—social media archives, note-taking applications, code repositories, fitness trackers, and more. They leverage the core VDFS abstractions to represent external data as native Spacedrive objects, seamlessly integrating it into the user's library.

Example Use Cases: A Unified Digital Life. The potential applications create a powerful, interconnected data ecosystem:

- **Social Media Archiver**: An agent for Twitter could import a user's entire tweet history. Each tweet becomes an `Entry`, its text content indexed for search, and its metadata (likes, retweets) stored in a sidecar.
- **Note-Taking Connector**: A Notion or Evernote agent could sync all notes into the VDFS. This would allow a user to create links between a specific note and a set of project files stored locally, bridging the gap between knowledge management and file management.
- **Location Tracker**: An agent connected to a location tracking service could ingest a user's location history. The AI could then use this data to automatically tag photos and videos with the location where they were captured.
- **Data-Driven Mini-Apps**: The architecture supports the creation of entire applications that use Spacedrive as their private, cross-device backend. A time-tracking app, for instance, could store all its data directly within the user's Spacedrive Library, gaining automatic synchronization and AI capabilities for free.

This model fosters an ecosystem of "mini-apps" that offload the hard problems of storage, synchronization, and intelligence to the Spacedrive core. It fulfills the ultimate vision of the VDFS: to provide a single, private, and intelligent foundation for a user's entire digital existence, far beyond the boundaries of the traditional filesystem.

A Security and Privacy Model

Key Takeaways

- **Defense in Depth**: SQLCipher database encryption + ChaCha20 network keys + TLS 1.3 transport = multi-layered protection
- **Battle-Tested Security**: Protection against real attacks: NAS compromise, stolen devices, cloud breaches, and insider threats

Spacedrive's architecture prioritizes user privacy and data security through end-to-end encryption, secure credential management, and a well-defined threat model designed for personal data scenarios.

A.1 Data Protection at Rest

All sensitive user data is encrypted using industry-standard cryptographic protocols:

A.1.1 Library Database Encryption. Each '.sdlibrary' directory employs transparent database encryption:

Library databases employ SQLCipher for transparent encryption at rest. Encryption keys are derived from user passwords using PBKDF2 with 100,000+ iterations and unique per-library salts. The unlocking process involves reading the salt, deriving the key, opening the encrypted database connection, and verifying access through a test query.

Key derivation: User passwords are strengthened using PBKDF2 with 100,000+ iterations and unique salts per library, providing strong protection against brute-force attacks.

A.1.2 *Network Identity Protection.*

Device cryptographic keys are stored encrypted in the enhanced device configuration:

Network identity protection employs a layered approach: Ed25519 private keys are encrypted using ChaCha20-Poly1305 with keys derived through Argon2id from user passwords. Public keys remain in plaintext for identity verification. Decryption involves deriving the key using Argon2id parameters and salt, then decrypting the private key data.

A.2 Network Security

All network communications employ end-to-end encryption with perfect forward secrecy:

A.2.1 *Iroh QUIC Transport Security.*

The Iroh networking stack provides multiple layers of security through secure connections that combine long-term device identity (Ed25519) with ephemeral session keys. Connection establishment involves a three-phase process: QUIC handshake with TLS 1.3, mutual device identity verification, and application-level key exchange for perfect forward secrecy.

Transport Layer Security: QUIC provides TLS 1.3 encryption for all network traffic, ensuring confidentiality and integrity.

Application Layer Security: Additional encryption using ephemeral keys provides perfect forward secrecy—compromising long-term device keys cannot decrypt past communications. This is particularly important for Spacedrop transfers, where each session uses completely ephemeral ECDH key exchange, ensuring that even if device keys are later compromised, past file transfers remain secure.

A.3 Credential Management

Spacedrive employs a secure credential storage system for cloud service integration:

Secure Credential Vault Architecture

Spacedrive implements a multi-layered credential protection system for cloud service integration:

Master Key Derivation

- User password transformed into cryptographically strong master key using PBKDF2
- Unique salt per credential vault prevents rainbow table attacks
- Key stretching with 100,000+ iterations provides brute-force resistance

Individual Credential Protection

- Each credential encrypted separately using ChaCha20-Poly1305 authenticated encryption
- Unique random nonce for each encryption operation ensures semantic security
- Detailed metadata tracking: service name, creation time, last access

Storage and Lifecycle Management

- Encrypted credentials stored in secure key-value mapping by service name
- Automatic timestamp tracking for security auditing and credential rotation
- Zero plaintext credential storage—everything encrypted at rest

Platform Integration: On supported platforms (macOS Keychain, Windows Credential Manager, Linux Secret Service), credentials are additionally protected by the OS credential store.

A.4 Threat Model

Spacedrive's security design addresses the following threat scenarios:

A.4.1 *Local Device Compromise.*

Threat: Unauthorized physical access to user device.

Mitigation: - Database encryption renders '.sdlibrary' directories unreadable without password - Network keys encrypted separately, requiring password for decryption - No plaintext credentials stored on disk

A.4.2 *Network Eavesdropping.*

Threat: Passive monitoring of network communications.

Mitigation: - All communications encrypted with TLS 1.3 via QUIC - Perfect forward secrecy prevents retroactive decryption - Device fingerprints prevent MITM attacks during pairing

A.4.3 *Cloud Service Compromise.*

Threat: Breach of connected cloud storage providers.

Mitigation: - Spacedrive never stores user data in cloud services—only metadata indices - Cloud credentials encrypted locally, not shared with Spacedrive services - Content addressing enables detection of tampered files

A.4.4 *Malicious Spacedrive Instance.*

Threat: Compromised or malicious Spacedrive installation.

Mitigation: - Libraries are portable and can be moved between trusted installations - Audit logs provide complete history of all operations - Action preview system prevents unauthorized operations

A.4.5 *Practical Attack Scenarios.*

To illustrate the robustness of Spacedrive's security model, consider these realistic attack scenarios:

Scenario 1: NAS Compromise and File Replacement

Attack: An attacker gains access to a user's NAS and replaces legitimate files with malicious versions, attempting to propagate malware across the user's device ecosystem.

Spacedrive Defense:

- Content addressing via BLAKE3 hashes immediately detects file modifications—the replaced files will have different hashes than the indexed versions
- The integrity verification system flags discrepancies during the next scan, alerting the user to potential tampering
- Version history tracking shows the exact timestamp of unauthorized modifications
- Quarantine mechanisms prevent automatic synchronization of suspicious files to other devices
- The audit log creates a forensic trail showing which files were modified and when

Scenario 2: Stolen Laptop with Sensitive Photo Library

Attack: A laptop containing a Spacedrive library with sensitive personal photos is stolen. The attacker attempts to access the photo collection and extract metadata about locations and people.

Spacedrive Defense:

- SQLCipher encryption on the library database prevents access without the user's password
- Photo metadata and AI-generated embeddings remain encrypted at rest
- Even with physical disk access, the attacker cannot: - View photo thumbnails (encrypted in cache) - Access location data from EXIF metadata (encrypted in database) - Extract face recognition data or object detection results (encrypted embeddings)
- The 100,000+ iteration PBKDF2 key derivation makes brute-force attacks computationally infeasible

Scenario 3: Compromised Cloud Storage Credentials

Attack: An attacker obtains a user's cloud storage credentials through a phishing attack and attempts to inject malicious files into the user's Spacedrive-managed cloud volumes.

Spacedrive Defense:

- Spacedrive's credential vault remains secure—the attacker only has cloud credentials, not the Spacedrive master password
- Content validation during cloud synchronization detects unexpected file additions
- The volume classification system isolates cloud storage from local trusted volumes
- File injection attempts are logged in the audit system with source attribution
- Users can revoke cloud volume access instantly without affecting local data
- Optional two-factor authentication on cloud volume operations provides additional protection

Scenario 4: Insider Threat in Collaborative Team

Attack: A disgruntled employee on a design team attempts to exfiltrate proprietary assets and delete project files before leaving the company.

Spacedrive Defense:

- RBAC system restricts the employee to their assigned role permissions—they may have "Contributor" access allowing edits but not bulk deletions
- The Action System's preview capability flags suspicious bulk operations for administrative review
- Every file access and operation is logged in the immutable audit trail with full attribution (user, device, timestamp)
- Data Loss Prevention (DLP) policies can detect and block unusual download patterns or transfers to external devices
- Time-based access controls automatically revoke permissions at employment end date
- The planned undo capability would allow administrators to instantly reverse any destructive actions
- Cryptographic device attestation ensures actions can only originate from company-managed devices

Scenario 5: Supply Chain Attack on Enterprise Deployment

Attack: An attacker attempts to compromise an enterprise Spacedrive deployment by injecting malicious code into a third-party integration or storage driver.

Spacedrive Defense:

- Containerized deployment isolates each component with strict network policies
- All actions flow through the centralized Action System, preventing direct database manipulation
- Cryptographic signatures on all deployed components ensure integrity
- The audit system's append-only design prevents log tampering to hide malicious activity
- Storage abstraction layer validates all operations against expected patterns
- Regular security scanning of container images and dependencies
- Option for air-gapped deployment in high-security environments

A.5 Certificate Pinning and API Security

A.5.1 Cloud Provider Certificate Pinning. Spacedrive implements robust certificate pinning for all cloud storage provider connections:

Implementation Strategy:

- Pin both leaf certificates and intermediate CA certificates for major providers
- Maintain backup pins for certificate rotation scenarios
- Implement graceful fallback with user notification if pins fail
- Regular updates through secure channels for pin refreshes

Provider-Specific Handling:

- **Google Drive**: Pin GTS root and intermediate certificates
- **Dropbox**: Pin DigiCert certificates with rotation monitoring
- **OneDrive**: Pin Microsoft PKI infrastructure certificates
- **S3-Compatible**: User-configurable pins for self-hosted instances

A.6 Rate Limiting and Abuse Prevention

A.6.1 Multi-Layer Rate Limiting Architecture. Spacedrive implements intelligent rate limiting to prevent abuse while maintaining performance:

API Rate Limiting:

- Per-device token bucket algorithm with configurable rates
- Separate limits for read operations (1000/min) and write operations (100/min)
- Exponential backoff for repeated limit violations
- Priority queuing for critical operations during limit conditions

Network-Level Protection:

- Connection rate limiting per IP address (10 new connections/minute)
- Bandwidth throttling for suspected abuse patterns
- Automatic blacklisting for persistent violators
- DDoS mitigation through connection pooling limits

Operation-Level Safeguards:

- Bulk operation limits (max 1000 files per action)
- Concurrent job restrictions based on device capabilities

- Smart scheduling to prevent resource exhaustion
- User-configurable limits for shared libraries

A.7 Audit Log System

The audit log system provides comprehensive tracking of user actions across all devices:

Action Tracking:

- Each action creates a detailed audit entry
- Cross-device synchronization of audit logs
- Sequence-based ordering for chronological integrity
- Export capability for external audit systems

Implementation Details:

```rust
pub struct AuditEntry {
    pub id: Uuid,
    pub timestamp: DateTime<Utc>,
    pub action: ActionType,
    pub device_id: DeviceId,
    pub details: serde_json::Value,
    pub sequence_number: u64,
}

impl AuditEntry {
    pub fn new(action: ActionType, device_id: DeviceId)
     -> Self {
        Self {
            id: Uuid::new_v4(),
            timestamp: Utc::now(),
            action,
            device_id,
            details: serde_json::Value::Null,
            sequence_number: 0, // Assigned during sync
        }
    }
}
```

Listing 11: Audit log entry structure

Sync Integration:

- Audit logs sync as UserMetadata across devices
- Sequence numbers ensure proper ordering
- Conflict resolution preserves all audit entries
- Background reconciliation maintains consistency

A.8 Spacedrive Cloud Service Privacy Model

The managed Spacedrive Cloud Service treats privacy as a fundamental design principle, not an afterthought:

A.8.1 End-to-End Encryption Architecture. The Cloud Core instance runs the standard Spacedrive software with no special privileges or backdoors:

Cryptographic Isolation:

- Each Cloud Core runs in an isolated container with unique cryptographic identity
- Network policies enforce that only paired devices can communicate
- No shared infrastructure between different user instances
- Complete data isolation at storage, network, and compute layers
- Data is encrypted in transit and at rest

A.8.2 Operational Security. Infrastructure access is strictly controlled and audited:

Administrative Access:

- No direct access to user containers or data volumes
- Administrative operations limited to resource management and health monitoring
- All infrastructure access logged and audited
- User data remains encrypted even during backup operations

Data Retention and Deletion:

- User data is permanently deleted within 30 days of account closure
- Cryptographic erasure ensures data cannot be recovered
- Users can export their entire library before deletion
- No data mining or analysis of user content

A.9 Privacy-Preserving AI

The AI-native architecture maintains privacy through multiple mechanisms:

Flexible AI Provider Selection for Privacy Control

Spacedrive supports multiple AI deployment models to balance privacy, performance, and capability:

Local AI Processing (Maximum Privacy)

- Integrates with Ollama for completely local AI model execution
- User data never leaves the device—complete privacy preservation
- Configurable endpoint and model selection for different AI capabilities
- Works offline once models are downloaded

Self-Hosted Solutions (Organizational Control)

- Custom AI infrastructure under user or organization control
- Flexible authentication options for enterprise deployment
- Complete control over data processing and model selection
- Ideal for organizations with specific privacy or compliance requirements

Cloud AI Services (Enhanced Capabilities)

- Access to state-of-the-art models from major AI providers
- Encrypted API key storage with detailed privacy policy tracking
- Transparent data processing terms presented to users for informed consent
- Metadata-only transmission—file contents remain local

Local Processing: Default to local AI models (Ollama) that never transmit user data externally.

Metadata-Only Cloud Processing: When using cloud AI services, only file metadata (names, types, sizes) are transmitted—never file contents.

User Control: Complete transparency about which AI provider processes which data, with granular user control over privacy vs. capability trade-offs.

A.10 Balancing Privacy and Public Sharing

Spacedrive's security model accommodates both zero-knowledge privacy and public content sharing through its library-based architecture.

A.10.1 Per-Library Encryption Policy. Each library maintains independent encryption settings:

- **Private Libraries** (default): Full SQLCipher encryption at rest
- **Public Libraries** (opt-in): Unencrypted for web serving

- **Hybrid Libraries**: Encrypted with selective public locations

```rust
pub struct LibraryConfig {
    pub encryption: EncryptionMode,
    pub public_sharing: PublicSharingConfig,
}

pub enum EncryptionMode {
    /// Full encryption (default)
    Encrypted { key_derivation: Argon2id },
    /// No encryption (for public content)
    Unencrypted,
    /// Encrypted with public locations
    Hybrid { public_locations: Vec<LocationId> },
}

pub struct PublicSharingConfig {
    /// Which core serves public content
    pub hosting_core: CoreIdentity,
    /// Custom domain (if any)
    pub custom_domain: Option<String>,
    /// Access control rules
    pub access_rules: Vec<AccessRule>,
}
```

Listing 12: Library encryption configuration

A.10.2 Secure Public Sharing Workflow. Users can share content publicly without compromising private data:

(1) Create a dedicated public library or location
(2) Configure which core hosts public content (cloud or self-hosted)
(3) Move/copy files to public locations
(4) Share generated URLs with recipients
(5) Private libraries remain fully encrypted throughout

A.10.3 Implementation Considerations. This dual-mode approach ensures:

- **Clear Boundaries**: Users explicitly choose what becomes public
- **No Encryption Downgrade**: Private libraries cannot be converted to public
- **Audit Trail**: All public sharing actions are logged
- **Revocable Access**: Public files can be made private instantly
- **Hosting Flexibility**: Any core can serve public content with proper setup

Security Implications. The system maintains security through isolation:

- Public and private data never mix within a library
- Encryption keys are never exposed to hosting infrastructure
- Access tokens are scoped to specific libraries and operations
- Public URLs use capability-based security (unguessable paths)

By making encryption optional but enabled by default, Spacedrive provides flexibility for content creators and enterprises while maintaining strong privacy guarantees for personal data.

B Conclusion

Key Takeaways

- **Accessible Design**: Enterprise-grade distributed file management for all users
- **Production Proven**: Real implementation handling millions of files with sub-100ms response times validates every architectural decision
- **Future Ready**: Solid foundation for AI agents, federated learning, and evolving human-computer interaction

Spacedrive rethinks personal file management by making distributed systems capabilities accessible to individual users. Through our implementation, we have shown that personal data management can evolve beyond simple file browsers to become intelligent, distributed systems that respect user ownership while providing enterprise-level capabilities, embodying the principles of local-first software [?] and ubiquitous computing [?].

B.1 Key Contributions and Real-World Impact

This work transforms personal file management through five innovations: AI-native natural language operations, universal cross-device file addressing, immediate metadata for every file, domain-separated synchronization, and performance-aware deduplication.

The impact is practical. Users manage millions of files across dozens of devices through a single interface, eliminating storage waste while maintaining sub-second response times. Data remains portable, privacy is preserved through local-first design, and AI enhancement comes without sacrificing user control. Our Rust implementation validates these concepts at scale.

B.2 System Integration

These innovations work together to create powerful combined capabilities. The Library abstraction makes backup and migration trivial (copy a directory), while SdPath enables seamless operations across that distributed storage. Content addressing works transparently with the sync system to maintain deduplication relationships even as files move between devices. The Temporal-Semantic Search architecture leverages both the content addressing and metadata systems to provide semantic discovery at traditional keyword search speeds.

B.3 Validation in Production

Spacedrive's architecture has been validated through production implementation in Rust, demonstrating that these concepts work reliably in practice. The system handles millions of files across multiple devices while maintaining sub-second response times for user operations. The extensive test framework, including multi-process distributed testing, ensures that the complex interactions between networking, synchronization, and file operations remain stable across diverse deployment scenarios.

B.4 Future Work and Roadmap

The architectural foundation laid by Spacedrive opens concrete paths for near-term enhancements and long-term research directions. Our immediate roadmap focuses on extending the AI capabilities to support complex multi-step workflows, such as "organize all vacation photos by year and location, then create albums for each trip." This involves expanding the Action system to support workflow composition while maintaining the same security and reversibility guarantees. Parallel to this, we are developing intelligent content analysis pipelines that leverage both local and cloud models to automatically extract semantic information from files—identifying people in photos, extracting key topics from documents, and understanding relationships between files based on content rather than just metadata. The existing FFmpeg integration provides a foundation for planned media processing capabilities including on-demand transcoding, format conversion, and adaptive streaming support for seamless media playback across devices.

In the medium term, our research agenda includes three major initiatives. First, we are exploring federated learning approaches that would allow users to benefit from collective intelligence about file organization patterns while maintaining complete privacy—the system would learn from aggregate behaviors without ever exposing individual file information. Second, we are developing advanced storage tiering algorithms that combine AI predictions with real-time access patterns to automatically move files between fast local storage, slower archives, and cloud services based on predicted access likelihood and user-defined cost constraints. Third, we are investigating cross-Library content discovery mechanisms that would allow users to identify duplicate content across different Libraries (perhaps owned by family members or team members) while maintaining the strong isolation guarantees that make Libraries portable and secure. Additionally, we are expanding protocol support via community plugins, enabling deeper integration with enterprise storage systems like SMB/NFS and emerging decentralized protocols.

The longer-term vision extends Spacedrive beyond personal file management into a platform for personal AI agents that understand and manage all aspects of a user's digital life. By providing a complete, versioned view of a user's file history combined with rich semantic understanding, Spacedrive could enable AI assistants that truly understand context—not just current file state but how information has evolved over time. This temporal understanding, combined with the robust Action system, would allow AI agents to perform complex organizational tasks with confidence, knowing that all actions are reversible and auditable. The architecture's emphasis on user control ensures that as these AI capabilities grow more sophisticated, users retain ultimate authority over their data, with all AI operations remaining transparent, explainable, and reversible.

B.5 Limitations

While Spacedrive advances personal data management, it has boundaries. The single-device database model limits scalability beyond 100M files without sharding, potentially constraining extreme enterprise use. Mobile resource constraints may delay background indexing on low-power devices. The AI layer, while privacy-focused, requires capable hardware for local models, and cloud alternatives introduce latency. Finally, while offline-first, initial setup requires internet for device pairing in distributed scenarios.

These limitations inform our roadmap, ensuring future iterations maintain core principles while expanding capabilities.

B.6 Broader Implications

Spacedrive shows that user-centric applications can also be powerful enterprise systems. Through careful domain separation, a local-first security model, and an architecture built for scale, we have demonstrated that it is possible to build a single platform that serves the needs of both individual users and large organizations. The key insight is that by starting with a foundation of user empowerment and data sovereignty, we create a system that naturally scales up to enterprise requirements while maintaining the simplicity and control that individual users demand.

This approach suggests a path forward for personal computing that moves beyond the current model of data scattered across incompatible cloud services toward truly user-controlled, portable, and intelligent data management. The native cloud service model presented in Section 5 is a clear manifestation of this principle, proving that cloud convenience does not have to come at the cost of architectural integrity or user control. By treating the cloud as just another trusted peer, Spacedrive offers a viable hybrid model for the future of personal data. By treating personal data as a unified library rather than a collection of disconnected files, users gain both the simplicity of traditional file management and the power of modern distributed systems.

Spacedrive's architecture provides a robust foundation for the next generation of computing—one that bridges personal and enterprise needs seamlessly. Whether serving an individual creator, a small team, or a global enterprise, the platform delivers the same core promise: unified access to all data, intelligent assistance without sacrificing control, and a user experience that makes powerful capabilities feel effortless. This provides a new approach for how users and organizations interact with their digital assets at any scale.

We invite researchers, developers, and users to contribute to Spacedrive's open-source ecosystem at https://github.com/spacedriveapp/spacedrive, advancing the future of personal data management.

Acknowledgments

We thank the open-source community, particularly the developers of the Rust programming language and its ecosystem, including the Tauri, Iroh, Tokio and SeaORM projects, whose work provided the foundation for this research.

The authors acknowledge the use of generative AI tools for assistance in drafting and refining the prose of this whitepaper. The core architectural concepts, technical details, and design are the original work of the authors, who take full responsibility for the content and accuracy of this paper.

A Glossary of Terms

Core Concepts

Action: A pre-visualized, durable file operation that can be simulated before execution. Actions are the primary way users interact with files in Spacedrive.

CAS (Content-Addressed Storage): A storage system where data is identified by its content hash rather than location, enabling automatic deduplication.

Entry: The fundamental data unit in Spacedrive representing any filesystem entity (file or directory) with immediate metadata capabilities.

Library: A portable, self-contained `.sdlibrary` directory containing a complete Spacedrive database, configuration, and metadata for a user's data ecosystem.

Temporal-Semantic Search: Spacedrive's two-stage hybrid search architecture combining temporal-first filtering with vector-enhanced semantic discovery.

SdPath: Spacedrive's universal path abstraction that transparently addresses files across devices, volumes, and cloud storage.

VDFS (Virtual Distributed File System): A unified index of all user data across devices while keeping files in their original locations.

Technical Components

Content Identity: A unique identifier based on file content hash that tracks all instances of identical content across the Library.

CRDT (Conflict-free Replicated Data Type): A data structure that automatically resolves conflicts in distributed systems. Note: Spacedrive v1 attempted a custom CRDT implementation that proved overly complex. V2 replaced this with a simpler domain separation approach (see Section 4.5.1).

FTS (Full-Text Search): Traditional keyword-based search capability integrated into Spacedrive's Temporal-Semantic Search system.

Phantom Path: A special SdPath variant representing files that may not currently exist but are referenced in the Library index.

Virtual Sidecar System: A system for managing derivative data (e.g., thumbnails, OCR text, transcripts) associated with a file Entry. These sidecar files are stored within the Spacedrive Library and linked to the original file via the VDFS index, ensuring the original file is never modified.

Volume: Any storage location (local drive, network mount, cloud service) that Spacedrive can access and index.

OpenDAL: Open Data Access Layer, providing unified access to cloud storage services.

ContentKind: Semantic categorization system that groups files into 17 intuitive categories (Image, Video, Code, etc.) beyond traditional MIME types.

File Type System: Advanced multi-method file identification combining extension matching, magic byte detection, and content analysis with confidence scoring.

Architecture & Deployment

Daemon: A background service that hosts the Spacedrive core engine, providing persistent state management and enabling multiple clients to connect concurrently.

Embedded Mode: Deployment model where the core is directly linked into the application binary, used for mobile apps and standalone distributions.

IPC (Inter-Process Communication): The mechanism for client-daemon communication using Unix domain sockets or named pipes with a JSON-RPC protocol.

WASM Plugin: WebAssembly-based extension running in a sandboxed environment.

Integration: WASM plugin providing system integration (e.g., cloud storage).

Synchronization & Networking

Library Sync: Spacedrive's leaderless, peer-to-peer synchronization system. It uses a hybrid model: simple state-based replication for device-authoritative data (like the file index) and a lightweight, HLC-ordered log for shared metadata (like tags and collections). Each syncable model defines its own conflict resolution strategy, from implicit union merge for tag assignments to deterministic last-writer-wins for scalar values. This eliminates central coordinators and single points of failure while ensuring eventual consistency.

Hybrid Logical Clock (HLC): A distributed timestamping mechanism that combines physical time with a logical counter to create a total ordering of events across multiple devices. Used in Spacedrive to deterministically resolve conflicts for shared metadata without requiring a central coordinator

Iroh: The peer-to-peer networking library used for device discovery and secure communication.

P2P (Peer-to-Peer): Direct device-to-device communication without requiring a central server.

RPC (Remote Procedure Call): The mechanism for executing operations on remote devices in the Spacedrive network.

Spacedrop: Spacedrive's ephemeral file sharing protocol enabling secure, AirDrop-like transfers between any devices without requiring prior pairing or trust relationships.

Storage & Performance

Adaptive Hashing: Strategic content sampling for large files to maintain deduplication effectiveness without full-file hashing overhead.

Intelligent Storage Tiering: Automatic management of hot (frequently accessed) and cold (archival) data across different storage tiers.

Semantic Label: User-friendly volume names (e.g., "Jamie's MacBook") that persist across device reconnections.

StorageClass: The effective storage tier of a file, determined by taking the more restrictive class between a Volume's `PhysicalClass` and a Location's `LogicalClass`.

PhysicalClass: An automatically detected property of a Volume that reflects its physical capabilities (e.g., Hot for SSD, Cold for an archival cloud tier).

LogicalClass: A user-defined property on a Location that reflects its intended organizational purpose (e.g., marking a folder as an archive).

Volume Classification: Platform-aware categorization of storage devices to optimize performance and user experience.

File Formats & Standards

AVIF: A modern image format used for efficient thumbnail storage.

JSON: JavaScript Object Notation, used for configuration and data exchange.

BLAKE3: The cryptographic hash function used for content addressing, chosen for its speed and security.

SQLite: The embedded database engine powering Spacedrive's local-first architecture.

UUID: Universally Unique Identifier used for device and entry identification.

WebP: An image format used for efficient thumbnail compression.

Platform Acronyms

API: Application Programming Interface.

CLI: Command Line Interface.

CPU: Central Processing Unit.

GUI: Graphical User Interface.

NAS: Network Attached Storage.

OS: Operating System.

SMB: Server Message Block protocol for network file sharing.

SQL: Structured Query Language.

UI: User Interface.

AI-Related Terms

AI-Native Architecture: Spacedrive's design philosophy where AI capabilities are foundational rather than added features.

Ollama: An open-source platform for running large language models locally.

Semantic Search: Search capability that understands meaning and context rather than just matching keywords.

Vector Search: Search using mathematical representations of content meaning for semantic similarity matching.