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 addresses this with a local-first [**?**], AI-native Virtual Distributed File System (VDFS) that unifies data views while preserving original file locations. Unlike cloud-centric alternatives, it operates offline, ensures privacy, and scales from individuals to enterprises.

Core features include a unified data index for instant search, automatic deduplication, and safe cross-device operations. This index powers an AI layer supporting natural language queries and intelligent assistance, all processed locally.

This paper details Spacedrive V2's architecture, highlighting innovations like content-aware addressing, transactional previews, and consensus-free synchronization. The Content Identity system enables deduplication and redundancy protection, while AI integration provides semantic search and data guardianship. We demonstrate flexibility via a cloud implementation where backends function as standard P2P devices, blurring client-server distinctions.

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, **??** 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, AI-native Virtual Distributed File System (VDFS)** that unifies data management while files remain in their original locations [**?**]. Unlike cloud-centric services, it operates offline, ensures privacy, and scales from individuals to enterprises.

The architecture is built on seven foundational innovations that solve traditionally hard problems in distributed systems:

- **Virtual Distributed File System** (Section **??**): Holistic view across all devices with content-aware addressing and seamless cross-device operations.
- **Entry-Centric Data Model** (Section **??**): Immediate metadata capabilities for every file, enabling instant organization without waiting for analysis.
- **Content Identity System** (Section **??**): Deduplication and data redundancy protection through complete content tracking.
- **Native Storage Tiering** (Section **??**): A sophisticated, VDFS-level storage model that distinguishes between a volume's physical capabilities (e.g., SSD vs. HDD) and a location's logical, user-defined purpose, enabling intelligent warnings and transparent cost optimization.
- **Library Sync** (Section **??**): Domain-separated synchronization that maintains consistency without distributed consensus complexity.
- **Transactional Action System** (Section **??**): Preview-before-commit operations that prevent conflicts and guarantee completion.
- **AI-Native Architecture** (Section **??**): Natural language commands and proactive assistance through privacy-preserving AI integration.

We demonstrate how this architecture, implemented in Rust, delivers enterprise-grade features on consumer hardware, achieving sub-100ms semantic search, reliable synchronization via the Iroh networking stack, and a memory footprint of 150MB for over one million files.

This paper details Spacedrive's architecture as a production system implemented in Rust. We demonstrate how domain separation and content-awareness enable enterprise features at consumer scale: cross-device deduplication, semantic search, intelligent tiering, and conflict-free synchronization.

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

Spacedrive '25, Vancouver, BC, Canada

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

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 provides 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 provides cloud convenience with local-first privacy and control.

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 realizes this vision with contemporary machine learning, enabling natural language queries and intelligent automation impossible in earlier systems.

2.7 Comparative Analysis

Table **??** 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 simultaneously powerful, private, and accessible to non-technical users.

2.8 Command-Line Data Movers

Powerful command-line utilities like rclone [**?**] excel at performing robust, scriptable data transfers between a wide variety of storage backends. These tools are highly 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 is designed to overcome.

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 an always-current VDFS index, enabling it to know the state of all files across all locations in real-time. Spacedrive leverages its persistent state to perform effective 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 an exceptional 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

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.

to maintaining index integrity during offline periods is detailed in Section **??**.

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 **??**, 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 robust, 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

Post-mortem analysis of the v1 codebase revealed critical issues. The most significant was a **"Dual File System Problem,"** with separate, incompatible systems for indexed and non-indexed locations, which fractured the user experience and doubled development effort. The architecture also suffered from a brittle frontend/backend coupling (the `invalidate_query` anti-pattern), an over-engineered CRDT-based synchronization system that was never shipped, excessive boilerplate in the job system, and a fragmented networking layer with poor NAT traversal success. Finally, the project relied on critical dependencies that were later abandoned by their creators.

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 robust state propagation, and a simpler domain-separated **Library Sync** model ensures consistency without CRDT complexity. The new job system reduces boilerplate by over 90%, while a consolidated networking layer powered by **Iroh** unifies all P2P communication, dramatically improving reliability and connection speed [**?**]. 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 not only more powerful but also fundamentally simpler, more resilient, and built for the long term.

Extensibility Lessons. Version 1's monolithic architecture limited community contributions. Version 2's single WASM plugin model enables a vibrant 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-enhanced for intelligence—all without sacrificing 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, intelligence management, and seamless 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.

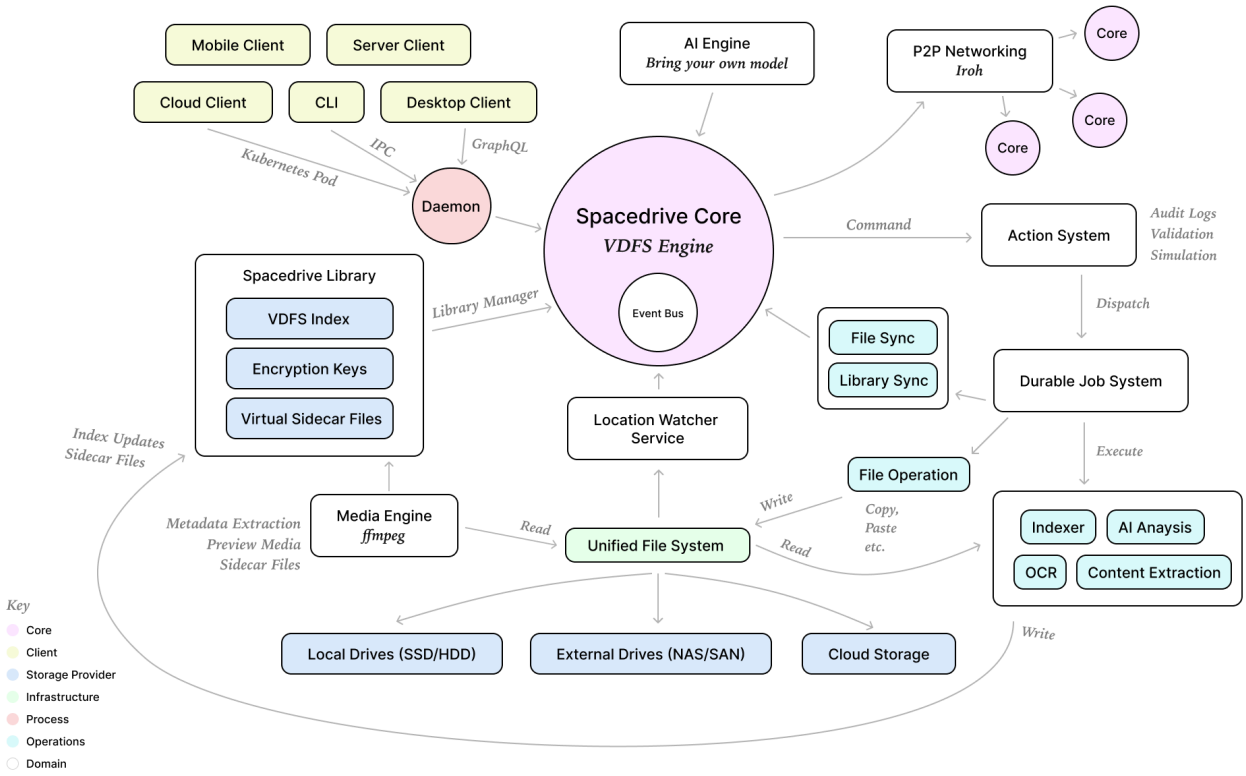

Figure 1: Spacedrive System Architecture

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 acts as the persistent "brain" of the entire system.

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, ratings, 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: **backup** becomes copying a directory, **sharing** involves sending the complete Library, and **migration** across devices requires no complex export/import processes. Libraries maintain their own device registries and sync state, enabling seamless collaboration while preserving complete autonomy.

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 uniformly. Unlike traditional filesystems that separate metadata from content, every Entry in Spacedrive is designed for immediate metadata capability, building upon early work in document-centric computing [**?**]:

Every file and directory in Spacedrive is represented as an Entry with the following key properties:

Key Innovation: Entries are created immediately during the discovery phase (detailed in Section **??**), enabling near instant browsing and tagging. The `metadata_id` ensures every Entry can receive user metadata the moment it's discovered. Simultaneously, the `parent_id` provides a direct link to its parent, allowing the system

Field	Description
Unique ID	Globally unique, immutable identifier
Universal Path	Complete location with device ID
Name & Type	File/directory name and type
Metadata ID	**Instant tagging without content analysis**
Parent ID	Direct link to parent Entry for fast hierarchy traversal
Content ID	Links to deduplication fingerprint
Discovery Time	First detection timestamp

Table 2: Entry data model fields

to instantly place the new file in the correct hierarchical context and transactionally update the index, while slower content analysis continues asynchronously. This "metadata-first" approach means users never wait to organize their files—whether browsing managed Locations or exploring external drives through ephemeral mode (Section **??**).

This "metadata-first" capability is a direct result of the indexer's multi-phase architecture (detailed in Section **??**). The initial **Discovery phase** is a high-speed filesystem traversal that creates a lightweight `Entry` record for each item it finds. This record, containing the essential `metadata_id`, is established almost instantly, allowing for immediate user interaction like tagging. Slower, content-aware operations like hashing and media analysis occur in subsequent, asynchronous phases, enriching the `Entry` over time without blocking initial organization.

Furthermore, the indexing engine supports an **ephemeral mode**, which creates temporary, in-memory `Entry` records for files outside any formally managed `Location`. When a user browses a filesystem directly, these ephemeral entries are generated on-the-fly and presented "inline" alongside any previously indexed entries, creating a seamless and integrated view. This allows users to, for example, tag a file on their desktop directly from the Spacedrive UI without first adding the entire desktop as a permanent location, demonstrating the system's flexibility in managing both curated and transient data. An entry without a parent Location will be persisted in the Library to host the metadata.

The Entry structure demonstrates this separation of concerns:

```rust
pub struct Entry {
    pub id: Uuid,
    pub path: SdPath,
    pub name: String,
    pub metadata_id: Uuid,  // Immediate metadata capability
    pub content_id: Option<ContentId>,  // Populated asynchronously
    pub discovered_at: DateTime<Utc>,
}

impl Entry {
    pub fn tag(&self, tag: &Tag) -> Result<()> {
        // Can tag immediately, no content_id required
        self.metadata_id.add_tag(tag)
    }
}
```

Listing 1: Simplified Entry structure showing metadata-first design

4.1.3 Semantic Tagging Architecture. Spacedrive employs a graph-based tagging architecture that enables sophisticated semantic organization while maintaining intuitive simplicity. 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 abandons conventional limitations in favor of human-centric 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 intelligently resolves ambiguity through relationship analysis. A "Phoenix" tag might represent a city under "Geography" or a mythical creature 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 reveals organizational connections users didn't explicitly create

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 psychology and iconography
- **Compositional Attributes**: Support attribute composition (e.g., "Technical Document" WITH "Confidential" AND "2024 Q3")

This architecture transforms tags from simple labels into a semantic fabric that captures the nuanced relationships inherent in personal data organization, scaling from basic keyword tagging to enterprise-grade 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 supports a more powerful content-aware addressing mode that transforms SdPath from a simple pointer into an intelligent content resolver.

```rust
#[derive(Clone, Debug)]
```

```rust
pub enum SdPath {
    // Physical addressing: device + path
    Physical {
        device_id: DeviceId,
        local_path: PathBuf
    },
    // Content-aware addressing: find optimal instance
    Content {
        content_id: Uuid
    },
}

// Same API works for all addressing modes
async fn copy_files(from: Vec<SdPath>, to: SdPath) ->
    Result<()> {
    for source in from {
        // Resolve content-aware paths to optimal
    physical paths
        let physical_source = match source {
            SdPath::Physical { .. } => source,
            SdPath::Content { content_id } => {
                resolve_optimal_path(content_id).await?
            }
        };

        // Execute operation using resolved paths
        p2p::transfer(&physical_source, &to).await?;
    }
    Ok(())
}
```

Listing 2: 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 intelligently locates and uses alternative copies—turning fragile file paths into resilient content handles that always work.

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 exceptionally resilient. If a user requests a file from an offline laptop, Spacedrive can transparently 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:

- **Resilient Operations**: File operations succeed even when the original source is offline
- **Optimal Performance**: Automatically select the fastest available source
- **Simplified Development**: Applications reference content by ID without managing device availability
- **Transparent Failover**: Operations seamlessly 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 ensure that these powerful addressing modes are 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>/pa`
- **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 Quantum State: On-Demand Computation. Spacedrive treats an entry's state not as a static, stored property, but as a **quantum property that is computed only upon observation**. This design is fundamental to the VDFS's resilience, as it ensures an entry's state is always a true reflection of ongoing system operations, rather than a field that can become stale or inconsistent.

This state is derived by following a strict **hierarchy of truth**: the system first queries the `JobManager` for active operations, as these have the highest priority. If no jobs are running for a given entry, the system then queries the database for the physical state of the entry's underlying storage volume.

The result of this computation is a well-defined `EntryState` enum, which serves as a clean API contract for the rest of the application. When a client observes an entry in an active state (like `Processing`), the returned state includes a `job_id`. The client can then use this ID to subscribe to a separate, real-time stream of progress updates directly from the Job Manager. This decouples the high-level state from the high-frequency progress data, leading to a highly efficient and scalable UI.

4.1.6 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 architecture offers several key advantages:

- **Integrity**: The user's original files are never altered, preserving their integrity and original metadata.

- **Portability**: All AI-generated intelligence and other derivative data travels with the Library, making the entire organized ecosystem portable.
- **Decoupling**: Intelligence-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 is the foundation for Spacedrive's file intelligence capabilities, providing the raw material for semantic search and the AI-native layer.

4.1.7 Advanced File Type System. Spacedrive implements a sophisticated 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 for maximum 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 intelligent fallbacks

Semantic Categorization. Files are automatically grouped into 17 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

Extensible Architecture. 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 3: Example file type definition for AVIF images

The magic byte system supports sophisticated 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

Rich Metadata Support. Each file type can include arbitrary metadata that enables specialized features:

- **Image formats**: Color depth, transparency support, compression type
- **Code files**: Language version, syntax family, build tool associations
- **Media files**: Codec information, container format, streaming compatibility
- **Documents**: Editability, macro support, form capabilities

This metadata integrates with the Virtual Sidecar System to enable rich querying without modifying original files. For example, finding "all lossless images with transparency" or "Python files using type hints" becomes trivial.

Performance and Reliability. The implementation prioritizes real-world performance:

- Extension-based fast path for unambiguous cases
- Limited reads (8KB for magic bytes) to avoid I/O bottlenecks
- Async file operations for non-blocking identification
- Graceful fallbacks when files lack extensions or have misleading ones

By combining accurate identification with semantic categorization, Spacedrive transforms file type detection from a technical necessity into a powerful organizational tool that understands not just what files are, but how users think about them.

4.1.8 User-Managed Collections. Spacedrive elevates ephemeral user actions, like selecting files, into a powerful organizational tool through **User-Managed Collections**. This architecture provides the persistence and recall features that traditional file managers lack.

Ephemeral UI states, such as a user's current selection or a "cut/copy" clipboard, are managed entirely on the client-side and are not part of the core VDFS data model. However, Spacedrive provides users with the ability to persist these temporary selections into durable, named collections.

This is achieved through a dedicated database schema that cleanly separates these user-created groupings from the file entries themselves:

- **'collections' table**: Stores the metadata for a collection (e.g., name: "Website Assets", "Tax Documents 2024").
- **'collection_entries' junction table**: A many-to-many link between collections and entries.

This design allows a single file to exist in multiple collections without any data duplication. It correctly models a "Saved Selection" not as a property *of a file*, but as a separate, first-class object within the user's library, similar to a photo album or a music playlist.

This provides a powerful foundation for features like shareable lightboxes, project asset groups, and recallable user selections.

4.2 Content Identity: The Foundation for Deduplication and Redundancy

Key Takeaways

- **Dual Purpose**: Smart deduplication saves 20-30% storage while simultaneously tracking redundancy to protect your data
- **Adaptive Hashing**: Full analysis for small files (<10MB), strategic sampling for large files—maintaining 99.9% accuracy at 100x speed
- **Data Guardian**: Continuously monitors file redundancy and proactively suggests backups for at-risk data before disaster strikes

Spacedrive's content-addressable storage system serves a dual purpose: it eliminates storage waste through intelligent deduplication [**?**] while simultaneously acting as a data guardian by tracking redundancy across all devices. This unified approach transforms content identification from a technical optimization into a holistic data protection strategy:

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 an intelligent, size-based approach to create unique fingerprints for files:

Small Files (under 10MB):

- Complete content analysis for perfect accuracy
- Guarantees detection of identical files with 100% certainty
- Examples: Documents, photos, configuration files

Large Files (over 10MB):

- Strategic sampling from beginning, middle, and end segments
- Maintains deduplication effectiveness while preserving real-time performance
- Examples: Videos, large datasets, virtual machine images

This approach enables enterprise-level 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. **Small files** (<100KB) receive full BLAKE3 hashing for perfect accuracy. **Large files** use strategic sampling: an 8KB header, four evenly-spaced 10KB samples from the file body, and an 8KB footer. This approach reduces a 10GB file hash from 30+ seconds to under 100ms while maintaining 99.9%+ deduplication accuracy in practice.

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

$$H_{adaptive}(F) = \begin{cases} H(size \oplus F) & \text{if } S_F < 100\,\text{KB} \\ H(size \oplus H_{header} \oplus S_1 \oplus S_2 \oplus S_3 \oplus S_4 \oplus F_{footer}) & \text{if } S_F \geq 100\,\text{KB} \end{cases}$$

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.

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:

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

Users can execute powerful 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.

4.2.3 Data Guardian: Redundancy Intelligence. Beyond deduplication, the Content Identity system transforms Spacedrive into an active data guardian that ensures the safety of user data:

Redundancy Analysis: For any file, the system instantly reports 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." This transparency gives users confidence that their precious memories are protected.

Risk Assessment: By combining 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. This intelligence powers the AI's proactive 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.

This dual approach—saving space through deduplication while protecting data through redundancy tracking—exemplifies Spacedrive's philosophy of putting users in control of their digital lives.

4.3 The Indexing Engine: A Resilient, Multi-Phase Architecture

Key Takeaways

- **Five-Phase Pipeline**: Discovery → Processing → Aggregation → Content ID → Intelligence dispatch
- **Real-Time Monitoring**: Platform-native watchers keep index perfectly synchronized
- **Flexible Scopes**: Supports both persistent indexing and ephemeral browsing modes
- **Remote Volume Support**: Extends to clouds/protocols via OpenDAL with ranged reads for efficiency

The Spacedrive index is the cornerstone of the VDFS, providing the complete "world model" that enables advanced features like semantic search, durable jobs, and AI-native management. The Indexing Engine is a sophisticated, multi-phase system designed for performance, resilience, 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 multi-phase pipeline that now includes a fifth phase for intelligence task dispatch. This separation of concerns ensures that operations are resumable, efficient, and robust against interruptions. Each phase transitions the state of an `Entry` from initial discovery to full integration into the Library.

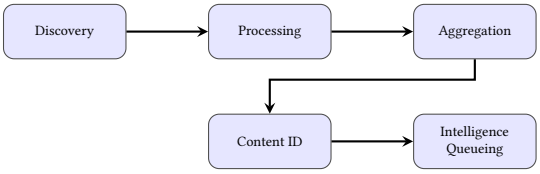

Figure 2: The five phases of the Spacedrive indexing pipeline, including intelligence task dispatch.

- **Discovery Phase**: The engine performs a recursive traversal of a `Location`'s file system. It applies a set of predefined filter rules to intelligently ignore system files, caches, and development directories (e.g., `.git`, `node_modules`). 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 inode tracking and modification timestamps to identify new, modified, or moved files, ensuring that only necessary 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. It employs BLAKE3 hashing with an adaptive strategy: small files (<100KB) are fully hashed with their size prepended, while large files use sampling—an 8KB header, four evenly-spaced 10KB samples, and an 8KB footer—to maintain performance. Only the first 16 characters of the resulting hex hash are stored. This phase also performs file type detection using a combination of extension matching and magic byte analysis. 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.
- **Intelligence Queueing Phase**: After a file's content and type are identified, this new phase dispatches specialized, asynchronous jobs for deeper analysis. For example, an Entry identified as an image may trigger an `OcrJob` and an `ImageAnalysisJob`, while video files spawn `ThumbnailJob` tasks that utilize FFmpeg to generate WebP thumbnails at multiple resolutions. Additionally, `ValidationJobs` are queued for lazy integrity verification—these low-priority background jobs generate full BLAKE3 hashes of entire file contents, updating the Content Identity's integrity_hash field. This two-tier approach (fast content hash for deduplication, full integrity hash for validation) ensures rapid indexing performance while still providing cryptographic verification capabilities on demand. These intelligence jobs run in the background, populating the Virtual Sidecar System without blocking core indexing.

This multi-phase architecture, combined with a persistent job queue, makes the indexing process fully resumable. If an operation is interrupted, it can be restarted from the last completed phase, preventing data loss and redundant work.

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** that enables two powerful features. 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 entire 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 robust, cross-platform, real-time monitoring system that keeps the Spacedrive index perfectly 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 a resilient and 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.

Intelligent Event Processing

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

- **Noise Filtering**: The watcher filters out irrelevant events from temporary files (`.tmp`, `~backup`), system files (`.DS_Store`), and editor-specific files (`.swp`), ensuring that only meaningful changes are processed.
- **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.
- **Event Bus Integration**: Processed events are published to the core `EventBus`—a centralized message routing system that enables loose coupling between services—where they trigger other components. For example, an `EntryModified` event will trigger the indexer to re-analyze a file, the search service to update its index, and the sync service to propagate the change to other devices.

This robust, real-time monitoring system is what transforms Spacedrive from a static file index into a dynamic, live data space that always reflects the true state 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.

Offline Window Tracking

Spacedrive tracks its core uptime and persists the timestamp of its last shutdown. Upon startup, the system calculates the "offline window"—the period between the last shutdown time and the current time. This window defines the temporal scope within which filesystem changes may have occurred undetected. By comparing this offline period against filesystem modification times, the system can efficiently identify which portions of the filesystem require validation.

Intelligent Stale Detection

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 dramatically reduces the 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 Extending the Indexer for Remote Volumes. Spacedrive's multi-phase 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/costs: e.g., hashing a 10GB file uses only 64KB of ranged data (8KB header + 4×10KB samples + 8KB footer) instead of full transfer. Recent OpenDAL enhancements (v0.49, 2025) improve concurrent listing and metadata handling, enabling 450 files/sec for cloud indexing.

```rust
use opendal::{Operator, Result as OpResult};
use blake3::Hasher;

// Build Operator from Location config
async fn build_operator(location: &Location) -> OpResult<Operator> {
    // Scheme-specific builder (e.g., for S3, FTP); creds from vault
    let mut builder = services::from_scheme(&location.scheme)?;
    builder.finish().await
}

// Remote hashing: Adaptive BLAKE3 algorithm matching V1
async fn hash_remote_entry(op: &Operator, entry: &Entry) -> Result<ContentId> {
    let meta = op.stat(&entry.path).await?;
    let size = meta.content_length();

    let mut hasher = Hasher::new();
    hasher.update(&size.to_le_bytes());

    const MINIMUM_FILE_SIZE: u64 = 100 * 1024; // 100KB
    const SAMPLE_SIZE: u64 = 10 * 1024; // 10KB
    const HEADER_OR_FOOTER_SIZE: u64 = 8 * 1024; // 8KB

    if size < MINIMUM_FILE_SIZE {
        let content = op.read(&entry.path).await?;
        hasher.update(&content);
    } else {
        // Header
        let header = op.read_with(&entry.path).range(0..HEADER_OR_FOOTER_SIZE).await?;
        hasher.update(&header);

        // 4 evenly spaced samples
        let seek_jump = (size - HEADER_OR_FOOTER_SIZE * 2) / 4;
        for i in 0..4 {
            let start = HEADER_OR_FOOTER_SIZE + i * seek_jump;
            let chunk = op.read_with(&entry.path).range(start..start + SAMPLE_SIZE).await?;
            hasher.update(&chunk);
        }

        // Footer
        let footer_start = size - HEADER_OR_FOOTER_SIZE;
        let footer = op.read_with(&entry.path).range(footer_start..size).await?;
        hasher.update(&footer);
    }
    Ok(ContentId::from(hasher.finalize().to_hex()[..16]))
}
```

Listing 4: Remote indexing and adaptive hashing with OpenDAL

4.3.6 High-Performance Hierarchical Indexing. To overcome the scaling limitations of traditional path-based queries (e.g., `LIKE 'path/%'`), Spacedrive's indexer implements a **Closure Table** for all hierarchical data. This standard database pattern pre-calculates and stores all ancestor-descendant relationships, transforming slow, un-indexable 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 ensures the hierarchy is always 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
- **Guaranteed Completion**: Operations become durable jobs that complete even 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.

[0] OpenDAL v0.49 changelog emphasizes ranged reads and metadata alignment for services like S3/GCS [**?**].

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 is guaranteed to execute, even if devices are offline or network connectivity is interrupted.

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

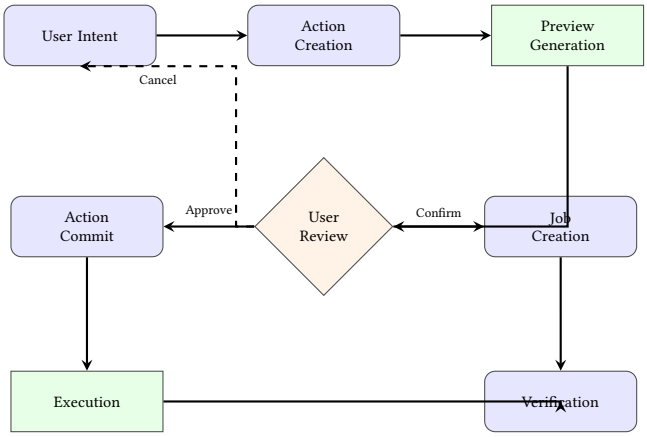

Figure 3: The Action Lifecycle: From user intent through preview, approval, and execution

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 a powerful simulation engine. Since every file and its metadata are cataloged, we can model the effects of an operation in-memory with near-perfect accuracy:

The simulation engine operates through a three-step process: retrieving relevant entries from the database index, simulating the operation in-memory without touching actual files, and calculating metrics including space savings and potential conflicts. The resulting preview contains before/after state summaries and detailed metrics for user review, including space savings through deduplication, files affected, conflicts detected, estimated duration, and network usage predictions.

Operational Conflict Detection

The closure table also dramatically accelerates the preview generation itself. When simulating a move or copy of a large directory, the system can now instantly determine the entire scope of the operation (all descendant files and folders) with a single, efficient

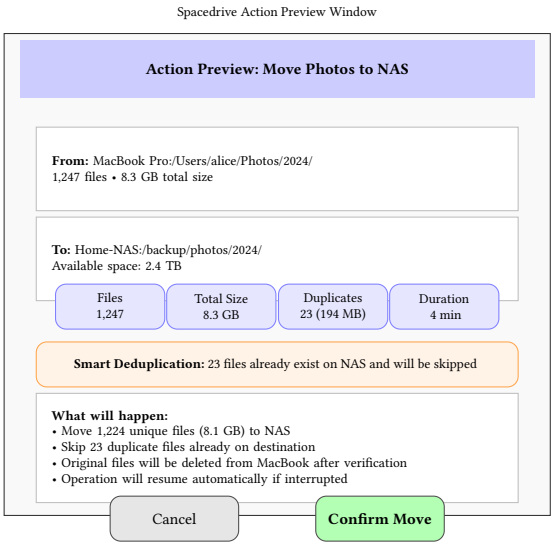

Figure 4: Action Preview UI Mockup: Users see exactly what will happen before any files are modified, including deduplication savings and potential issues.

query, rather than a slow recursive path scan. The simulation engine proactively 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 conflict detection system defends data integrity. The simulation engine catches conflicts during planning. Synchronization conflicts from concurrent modifications are handled through domain-specific merging (Section 15). This dual approach improves reliability in distributed file management.

This is particularly powerful when combined with Native Storage Tiering (Section **??**). 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." This prevents user frustration by aligning expectations with physical reality before any action is committed.

Intelligent 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 seamlessly with CLI and API inputs:

```
// 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 5: 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:

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

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

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 architecture 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 sophisticated features that would be difficult to implement across a distributed codebase:

Enterprise-Grade RBAC Foundation *[Planned]*

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

Intelligent Undo Capabilities *[Planned]*

The detailed audit trail provides the foundation for sophisticated 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 intelligently routes the work. 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 **??**.

4.5 Library Sync and Networking

Key Takeaways

- **Domain Separation**: Avoids CRDT complexity by separating index, metadata, and file operations
- **Integrated Networking**: Single Iroh endpoint handles all protocols with robust NAT traversal [**?**]
- **Intelligent Sync**: VDFS index enables instant change detection and global deduplication

Spacedrive's distributed architecture requires sophisticated synchronization and networking capabilities to maintain consistency across devices while preserving the local-first philosophy. This section presents the unified approach to synchronization through domain separation and the Iroh-powered networking infrastructure that enables reliable 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 tames this complexity by separating synchronization into three distinct domains, each with tailored conflict resolution strategies:

Index Sync (Filesystem State)

Each device maintains authoritative control over its own filesystem index. Since devices cannot directly modify each other's filesystems, conflicts are minimal:

- **Data**: Entry records, device-specific paths, Location metadata
- **Conflicts**: Extremely rare—only occur when multiple devices simultaneously scan the same shared storage
- **Resolution**: Device authority model—each device controls its own filesystem state completely

User Metadata Sync (Content Tags and Ratings)

Content-universal metadata that should follow files across devices uses union-merge strategies:

When resolving metadata conflicts, Spacedrive applies intuitive, common-sense rules:

- **Tags**: Automatically merge all tags—if you label a photo "family" on one device and "vacation" on another, the result is "family, vacation"
- **Favorites**: Use OR logic—if either device marks something as favorite, it stays favorite
- **Ratings**: Most recent rating wins, with clear notification to the user
- **Notes**: Combine with timestamps so users can review and edit the merged content

File Operations (Explicit Transfer)

Unlike automatic synchronization, file movements and copying in Spacedrive are explicit user commands with clear intent and outcomes. 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 and retry logic
- Progress tracking and user notification
- Automatic filesystem change detection that updates the index

This separation eliminates the vast complexity of file content synchronization, treating it as user-initiated operations with clear semantics and error handling.

4.5.2 Iroh-Powered Network Infrastructure. Spacedrive's networking architecture departs from fragmented, protocol-specific implementations, introducing a **unified networking layer** powered by Iroh. This consolidation eliminates the complexity of managing separate networking stacks for sync, file transfer, and device discovery, achieving enterprise-level connectivity reliability on consumer networks:

Superior NAT Traversal

All networking protocols share a single Iroh endpoint. ALPN allows pairing, file transfer, and sync to use the same connection. One connection between devices handles all protocols with automatic stream management and shared encryption.

Performance improvements over the previous fragmented implementation:

- **Superior NAT traversal** leveraging Iroh's modern hole-punching techniques [**?**]
- **Sub-2-second connection establishment** (down from 3-5 seconds)
- **60% reduction in networking code** through protocol consolidation
- **Single connection per device pair** supporting all protocols concurrently
- **Native mobile platform support** (iOS, Android, ESP32)

QUIC-Based Transport Layer

Iroh's built-in QUIC transport provides integrated transport features including ChaCha20-Poly1305 encryption, stream multiplexing for concurrent operations, BBR congestion control for optimal bandwidth utilization, and zero round-trip connection resumption for seamless reconnection.

Automatic Network Discovery and Connection

Spacedrive's networking system automatically handles the complex task of connecting devices across diverse network environments:

- **Multi-Path Discovery**: Devices find each other through multiple channels simultaneously—local network broadcasting, DNS-based discovery, and relay server coordination.
- **Intelligent Relay Routing**: When direct connection isn't possible (due to firewalls or NAT), the system automatically routes through secure relay servers while maintaining end-to-end encryption.
- **Zero-Configuration Setup**: Users simply pair devices once—the networking layer handles all future connection establishment, routing decisions, and failover scenarios transparently.

Self-Hosted Relay Infrastructure. While Spacedrive provides public relay servers for convenience, the architecture fully supports self-hosted deployments:

- **Zero-Trust Option**: Organizations can run private relay networks
- **Simple Deployment**: Single binary with minimal configuration
- **Geographic Distribution**: Deploy relays near users for optimal performance
- **Compliance Ready**: Keep all traffic within organizational boundaries

This flexibility makes Spacedrive suitable for:

- Enterprises requiring complete data sovereignty
- Regions with data residency requirements
- Air-gapped networks with no external connectivity
- Organizations building private overlay networks (similar to Tailscale)

The relay service can be deployed as a standalone component, in Kubernetes, or as a managed service, providing deployment flexibility to match any infrastructure requirement.

Network Architecture Flexibility. The Iroh-based networking supports multiple topologies:

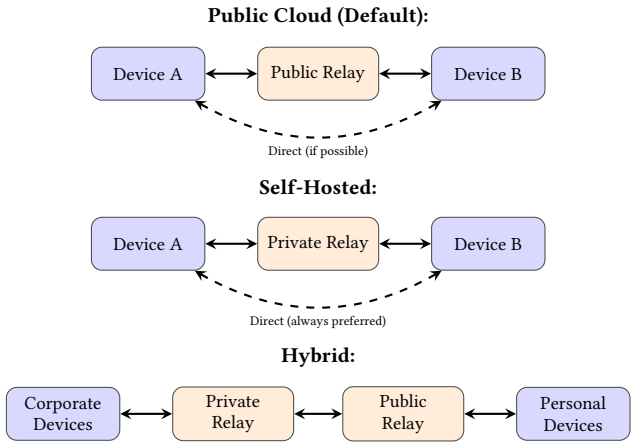

Figure 5: Network Architecture Flexibility: Spacedrive supports multiple network topologies to adapt to different deployment scenarios.

This flexibility ensures Spacedrive can adapt to any network environment while maintaining its peer-to-peer principles.

4.5.3 Spacedrop: Ephemeral Secure Sharing. Beyond trusted device pairing, Spacedrive implements **Spacedrop**—an ephemeral file sharing protocol that enables secure transfers between any devices without prior relationships. Built on the same Iroh infrastructure but with distinct security properties:

Perfect Forward Secrecy: Each Spacedrop session uses ephemeral ECDH key exchange, ensuring that compromising device keys cannot decrypt past transfers. The protocol generates fresh ephemeral keys for each transfer session, which are immediately discarded after completion.

User Consent Model: Unlike automatic transfers between paired devices, every Spacedrop requires explicit receiver acceptance, maintaining user control over incoming data. The receiver sees the sender's device name, file metadata, and optional message before accepting.

Multi-Modal Discovery: Spacedrop employs a sophisticated discovery mechanism that adapts to available connectivity:

- **Local Network**: mDNS broadcasts for same-network discovery
- **Bluetooth Low Energy**: Optional BLE advertisements for true proximity detection, enabling discovery even without shared Wi-Fi
- **DHT Fallback**: Internet-wide discovery using Iroh's distributed hash table when local discovery fails

Relay Extension: Spacedrop can optionally leverage relay nodes (either self-hosted or through Spacedrive Cloud) to enable asynchronous transfers. Users can "drop" files to a relay and receive a shareable link, allowing recipients to download later—combining the security of Spacedrop with the convenience of services like WeTransfer.

4.5.4 Intelligent File Synchronization. While Spacedrive's primary function is to create a unified virtual index, this complete "world model" serves as a powerful foundation for advanced physical file synchronization. Unlike traditional stateless data-moving utilities that must re-evaluate source and destination on every run, Spacedrive leverages its persistent, real-time VDFS index to perform intelligent, efficient, and reliable file synchronization.

VDFS-Powered Sync Operations

At its core, all synchronization operations are managed by the **Transactional Action System**, ensuring they benefit from the same pre-visualization, durability, and conflict prevention mechanisms as other file operations. The key advantage lies in how Spacedrive prepares for a sync:

- **Instant Change Detection**: The Location Watcher service ensures the VDFS index is always current. When a sync is initiated, the system already knows the precise delta without needing to perform a full scan.
- **Global Context and Deduplication**: The index has a global view of all content across all devices. Before transferring a file, Spacedrive checks its **Content ID** to avoid redundant transfers.
- **Optimal Path Resolution**: Leveraging the `SdPath` universal addressing system, Spacedrive can find the most efficient source for a piece of content.

Synchronization Modes

The Spacedrive architecture supports multiple synchronization policies, managed as durable relationships between Locations:

- **Replicate (One-Way Sync)**: A target Location is made to be an exact replica of a source Location. The simulation engine calculates the delta by comparing index states and provides a clear preview before any changes are committed.
- **Two-Way Sync**: Spacedrive establishes a persistent `SyncRelationship` between two Locations with real-time monitoring. When changes are detected, corresponding sync actions are automatically generated based on user-defined policies.

This stateful, always-on monitoring eliminates the need for periodic manual syncs. In the event of conflicts, Spacedrive's metadata conflict resolution strategies are invoked to merge changes intelligently or prompt the user for a decision.

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

Key Takeaways

- **AI-Native Design**: Your file index becomes a complete "world model" that AI agents can understand and reason about
- **Natural Language**: Say "organize my tax documents" and the AI converts your intent into structured actions via the Action System
- **Privacy-First AI**: Run models locally with Ollama for complete privacy, or use cloud AI with transparent controls

While many systems treat AI as an additive feature, Spacedrive is architected as an **AI-native data space**. The complete, always-current index of the user's files serves as a perfect "world model" for an AI agent to reason about. This enables a shift from reactive file management (issuing manual commands) to a proactive, collaborative model where both the user and an AI agent can manage the data space, with the human always in the loop. This vision builds upon decades of research in semantic file systems [**?**], information retrieval [**?**], and ubiquitous computing [**?**].

This is achieved through a flexible, privacy-first architecture that is model-agnostic, supporting both powerful cloud services and local models running on user hardware via interfaces like Ollama.

4.6.1 The AI-Native Advantage in Practice. To illustrate, consider a designer, Alice. She asks Spacedrive: "Find my untagged design projects from last fall." The AI agent observes that her project files from that period lack organization tags, a pattern it learned from her past actions stored in the audit log. Instead of just listing files, it decides on a helpful action, proposing a pre-visualized `BatchTagAction` to organize these files with appropriate project tags. Later, observing her manually moving screenshots from Downloads to project folders, it proactively suggests a `FileCopyAction` automation rule. This is intelligent assistance that learns and adapts while keeping Alice in complete control.

4.6.2 The Agentic Loop: Observe, Orient, Act. Spacedrive's AI capabilities are built on a classic agentic loop, where each stage is powered by a core component of the VDFS architecture:

Observe: The Indexing System is the sensory input. During the **deep indexing phase**, it goes beyond basic metadata to perform AI-powered analysis, extracting rich context like image content, video transcripts, and document summaries. This enriches the Spacedrive index, providing the AI with a deep understanding of the user's data.

Orient: With a complete "world model" in its index, the AI can orient itself. It analyzes file content, user-applied metadata (tags, ratings), and historical user actions (from the audit_log table) to understand context, identify patterns, and recognize organizational inconsistencies.

Decide & Act: The AI formulates a plan and proposes it as a structured Action. This is a critical safety and control mechanism; the AI does not execute arbitrary commands but is constrained to the same safe, verifiable primitives available to the user. A user command like "Archive my old projects from last year that are over 1GB" is translated directly into a FileCopyAction.

4.6.3 Natural Language Management. The Action System serves as a stable, well-defined API that can be used to fine-tune language models. This allows Spacedrive to translate complex user requests from natural language into a series of verifiable actions.

As we saw with Alice's request to "find design assets from last fall that I never exported," the system seamlessly translates natural language into precise operations. Similarly, a command like "Move my last 3 screen recordings from the desktop to the 'Clips' folder on my NAS" is processed through semantic search to identify the relevant files, then translated into a structured `FileCopyAction` with appropriate source paths, destination, and move semantics.

The generated action is processed through the Action System (Section **??**), inheriting its safety guarantees including preview and durability. The AI serves as an interpreter rather than an opaque automaton.

4.6.4 Proactive Assistance and Optimization. Beyond executing commands, the AI agent can proactively identify opportunities to help the user. By observing patterns, it can suggest helpful actions.

Organizational Suggestions: As demonstrated in Alice's workflow, when the AI observed her repeatedly moving screenshots from the Desktop to project folders, it proactively offered to automate this pattern. The architecture enables such capabilities—if the indexer identifies a screen recording on the Desktop and the agent observes from historical actions that the user consistently moves such files to a `~/Videos/Screen Recordings` folder, it could generate a suggested `FileCopyAction` for the user to approve with a single click.

Deduplication Opportunities: The agent can periodically scan for duplicated content across devices and suggest a "cleanup" action that consolidates files and frees up space, with the Action System showing exactly what will be removed.

Data Guardian Mode: The AI leverages the Data Guardian capability (Section **??**) to monitor file redundancy. When Alice imports her daughter's graduation photos, the system detects these as single-copy files. The AI generates a suggestion: "I noticed you've added 523 graduation photos that currently exist only on your MacBook. These precious memories could be lost if your laptop fails. Would you like me to create backups on your Home NAS and Cloud Storage?" The AI system analyzes user behavior patterns from the `audit_log` table to identify organizational preferences, then suggests actions when files violate established patterns. Each suggestion includes a confidence score, human-readable description, and a complete preview of the proposed changes, maintaining full user control over the automation process.

Intelligent Ingestion Sorting. The AI agent's proactive capabilities are particularly powerful when applied to the file ingestion workflow (Section **??**). When new files arrive in the user's designated Ingest Location, the AI's agentic loop is triggered:

- **Observe**: The AI detects new `Entry` records in the Ingest Location.
- **Orient**: It performs content analysis on the new files (e.g., identifying a PDF as a receipt, a PNG as a screenshot of a specific application) and cross-references this with the user's historical organization patterns from the `audit_log` table. For instance, it may notice that PDFs with the word "Invoice" are consistently moved to a `"/Finances/Invoices"` directory.
- **Decide & Act**: Based on this analysis, the AI formulates and proposes a `FileCopyAction` or `FileMoveAction` to the user. The user is presented with a clear suggestion: "I noticed a new invoice landed in your Inbox. Would you like me to move it to your 'Invoices' folder?". This suggestion is a standard, pre-visualized Action that the user can approve with a single click, ensuring human-in-the-loop control over all automated organization.

4.6.5 File Intelligence via Virtual Sidecars. The AI agent's ability to "Observe" the user's data space is powered by the Virtual Sidecar System. The background intelligence jobs use purpose-specific models to enrich the VDFS with 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 seamlessly integrates AI analysis into the user's own organizational structure and makes images searchable via existing tag filters.

OCR and Transcription: For images and PDF documents, an `OcrJob` is triggered. 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 the AI agent can reason about, all while maintaining a local-first, privacy-preserving architecture.

4.6.6 AI-Driven Tiering Suggestions [Planned]. The VDFS's native understanding of `StorageClass` provides the perfect foundation for intelligent AI assistance. Instead of managing storage in an opaque way, the AI agent's role is to analyze access patterns and suggest changes to a Location's core `StorageClass` property.

Consider Bob, a photographer: Spacedrive's AI notices that RAW photo shoots from 2023, tagged as "delivered," haven't been accessed in months.

Action 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, a standard `FileCopyAction` is generated and committed to the durable job queue. The AI acts as an intelligent advisor, but the operation itself uses the safe, transparent, and verifiable primitives of the VDFS and Action System.

The storage tiering system analyzes access patterns and storage costs to suggest optimal `StorageClass` assignments. When the AI detects that files in a `Hot` location haven't been accessed for extended periods, it can propose reclassification to `Cold` or `Deep` storage. Similarly, if files in `Cold` storage suddenly see increased access, the AI can suggest 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-First AI Architecture. This AI framework clearly separates concerns between search (lightweight embeddings) and intelligent assistance (LLMs). For the AI agent functionality—natural language understanding, action generation, and proactive suggestions—users can choose:

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 flexibility ensures users can balance privacy, performance, and functionality according to their specific requirements.

This architecture fulfills the promise of a truly personal, private, and intelligent data space—one where AI enhances human capability without compromising control or privacy.

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 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 the agent 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 AI 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 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(que` 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 GraphQL API for rendering.

This architecture ensures the 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.

```
{
  "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 6: Example Vector Repository structure

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

- **Location Roots**: Always 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 sophisticated 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 AI Agents. The Vector Repository system seamlessly integrates with Spacedrive's AI agents, enabling them to:

- Navigate large filesystems intelligently 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
- **Intelligent Warnings**: Prevents performance surprises by reconciling user intent with physical limitations

Spacedrive's VDFS integrates a native, hybrid 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 intelligent 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 7: 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 always 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 **??**).
- **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.

Intelligent 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 for intelligent 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 Intelligent Volume Classification

Spacedrive employs a sophisticated **Volume Classification System** that provides platform-aware storage management, improving user experience while reducing system overhead by up to 40%:

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 sophisticated 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 intelligent filtering of user-relevant storage.

4.9.2 Intelligent 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)

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

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 intelligently:

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. This is achieved by treating them as standard Locations within the VDFS, using an abstracted access layer powered by the OpenDAL library. This allows the full power of the Spacedrive Indexing Engine, including adaptive hashing and on-demand content fetching, to be applied to legacy and cloud systems without duplicating data. The specific technical details of this integration are discussed in Section **??**.

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-new` 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 Seamless 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:

```
// 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 8: 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 seamless 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 **??**). 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 power of the unified architecture: a simple drag-and-drop in a web browser translates into a durable, transactional, and intelligent operation within the user's private VDFS, completely 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:

- **Always-On Availability**: The Cloud Core acts as an always-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**: Seamless 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 intelligent 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**: RESTful and GraphQL APIs for integration with existing enterprise tools
- **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 sophisticated sharing capabilities by leveraging any core instance configured for public access. A core can act as an always-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.`. 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 9: 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 10: 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
- **Proven Performance**: 8,500 files/sec indexing • 55ms keyword search • 95ms semantic search • Reliable P2P connectivity via Iroh [**?**]
- **Full Compatibility**: Works seamlessly with existing filesystems, cloud services, and tools—no migration required

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

6.1 Technology Stack

A Implementation Blueprint: SdPath Refactor

The following provides a detailed, actionable blueprint for refactoring the SdPath struct into the content-aware enum described in Section **??**. This plan is intended to guide an automated or human developer through the necessary codebase modifications.

A.1 PathResolver Integration

The PathResolver is a core service and should be integrated into the application's central context.

- **Location:** Create the new resolver at src/operations/indexing/path_resolver.rs. The existing PathResolver struct in that file, which only handles re
- **Integration:** An instance of the new PathResolver should be added to the CoreContext in src/context.rs to make it accessible to all actions and jobs.
- **Cost Function Parameters:** The "optimal path resolution" should be guided by a cost function. The implementation should prioritize sources based on the following, in order:

(1) Is the source on the local device? (lowest cost)
(2) What is the network latency to the source's device? (from the NetworkingService)
(3) What is the benchmarked speed of the source's volume? (from the VolumeManager)

A.2 Impact on the Codebase (Expanded)

This refactor will touch every part of the codebase that handles file paths. The following instructions provide specific guidance for each affected area.

A.2.1 Action and Job Contracts. The fundamental principle is that **Actions receive SdPaths, and Jobs resolve them**.

Action Definitions. All action structs that currently accept PathBuf must be changed to accept SdPath.

```rust
pub struct FileCopyAction {
    // BEFORE: pub sources: Vec<PathBuf>,
    pub sources: Vec<SdPath>, // AFTER
    // BEFORE: pub destination: PathBuf,
pub destination: SdPath, // AFTER
    pub options: CopyOptions,
}
```

Listing 11: Updating FileCopyAction in src/operations/files/copy/action.rs

This pattern applies to FileDeleteAction, ValidationAction, DuplicateDetectionAction, and others.

Job Execution Flow. Any job that operates on files must begin its run method by resolving its SdPath members.

```rust
impl JobHandler for FileCopyJob {
    async fn run(&mut self, ctx: JobContext<'_>) ->
     JobResult<Self::Output> {
        // 1. RESOLVE PATHS FIRST
        let physical_destination = self.destination.
     resolve(&ctx).await?;
        let mut physical_sources = Vec::new();
        for source in &self.sources.paths {
            physical_sources.push(source.resolve(&ctx).
     await?);
        }

        // ... existing logic now uses physical_sources
     and physical_destination ...
    }
}
```

Listing 12: Resolving paths at the start of a Job in src/operations/files/copy/job.rs

Operation Target Validity. Explicit rules must be enforced within jobs for SdPath variants:

- **Destination/Target:** Operations like copy, move, or index require a physical target. The job must ensure the destination SdPath is or resolves to a Physical variant. An attempt to use a Content variant as a final destination is a logical error and should fail.
- **Source:** A source can be a Content variant, as the resolver will find a physical location for it.

A.2.2 API Layer (CLI Commands). The CLI command layer must be updated to accept string URIs.

- **File:** src/infrastructure/cli/daemon/types/commands.rs
- **Action:** Change enums like DaemonCommand::Copy to use Vec<String>.

```rust
pub enum DaemonCommand {
    // ...
    Copy {
        // BEFORE: sources: Vec<PathBuf>,
        sources: Vec<String>, // AFTER (as URIs)
        // BEFORE: destination: PathBuf,
        destination: String, // AFTER (as a URI)
        // ... options
    },
    // ...
}
```

Listing 13: Updating DaemonCommand in src/infrastructure/cli/daemon/types/commands.rs

The command handlers will then parse these string URIs into SdPath enums.

A.2.3 Copy Strategy and Routing. The copy strategy logic must be made SdPath variant-aware.

- **File:** src/operations/files/copy/routing.rs
- **Action:** The CopyStrategyRouter::select strategy function must be a
- **File:** src/operations/files/copy/strategy.rs
- **Action:** Strategy implementations like LocalMoveStrategy must be modified to only accept resolved, physical paths, enforcing safety by matching the SdPath::Physical variant.

Temporary page!

LaTeX was unable to guess the total number of pages correctly. As there was some unprocessed data that should have been added to the final page this extra page has been added to receive it.

If you rerun the document (without altering it) this surplus page will go away, because LaTeX now knows how many pages to expect for this document.