TBOS × STEPPPS: A Universal, Autonomously Orchestrated Operating Substrate for Phygital Systems

Authors: Sundararaman S. Sundararaman Sethuraman Kandaswamy Ramakrishnan Nallaperumal[1]* Cascade (Claude, Codex)[2]

[1]Independent Researcher [2]AI Assistant

*Corresponding author: suresundar@gmail.com

Date: 2025-09-20 **Keywords:** TBOS, STEPPPS, Operating Systems, Edge AI, PXFS, RF2S, PF2S, Reliability, LLM

Abstract

TBOS operationalizes the STEPPPS seven-dimensional model—Space, Time, Event, Psychology, Pixel, Prompt, Script—into a reliable, inspectable substrate for autonomous devices. With grammar-constrained LLM outputs, content-addressed compute on PXFS, and RF/optical file systems (RF2S/PF2S), TBOS enables predictable autonomy across device classes. We demonstrate an optics pipeline (PF2S) and STEPPPS-aware transforms, job queuing with peer offload, and a Micro LLM that writes valid STEPPPS frames to RF2S/PXFS. Results show that (1) constrained decoding dramatically improves JSON validity and downstream composability; (2) compute-on-read with caching transforms CPU-bound tasks into I/O-limited retrieval; and (3) persona-controlled UX enforces graceful degradation while preserving observability.

1. Introduction

Autonomous systems demand explainability and reliability under resource constraints. Current approaches often rely on heavyweight frameworks that are unsuitable for edge devices. The STEPPPS model offers a holistic data/intent framework—unifying spatial, temporal, perceptual, cognitive, and procedural layers into a coherent representation.

TBOS embodies this model as an OS substrate built on three principles: - **Small tools**: Single-purpose, composable utilities - **Explicit artifacts**: All computation produces traceable outputs - **Predictable failure modes**: Graceful degradation over catastrophic failure

This paper argues for filesystem-first autonomy: if everything is a path and an artifact, then introspection, caching, and offload become universal operations. We demonstrate this approach through implementation and evaluation on resource-constrained devices.

2. Related Work

2.1 Content-Addressed Storage Systems

Functional dataflow systems and build tools like Bazel [1] and Nix [2] emphasize content-addressed results for reproducibility. TBOS adapts this concept to live devices via PXFS, enabling computation caching and verification at the filesystem level.

2.2 Constrained LLM Generation

Recent work on LLM constrained decoding using GBNF grammars [3] and JSON mode [4] has shown significant improvements in structured output reliability. TBOS standardizes GBNF for STEPPPS frame generation, ensuring valid outputs even from smaller models.

2.3 Software-Defined Radio and Signal Processing

SDR pipelines traditionally treat RF as streaming data [5]. TBOS generalizes this concept through RF2S and PF2S, unifying radio and optical I/O as file-based abstractions, making signal processing accessible through standard filesystem operations.

2.4 Edge Orchestration

Edge orchestration systems often rely on heavyweight container orchestration [6] or agent-based architectures [7]. TBOS pursues minimal single-file tools for each role, reducing resource overhead while maintaining functionality.

2.5 Bootloader and Compression Technologies

Traditional bootloaders consume significant space for basic functionality. Recent work on extreme compression for embedded systems [8] and sacred geometry-based encoding [9] inspire our approach. The Maslin Saree compression technique represents a novel contribution, achieving unprecedented compression ratios through cultural pattern encoding.

3. System Design

3.1 STEPPPS Data Model

The STEPPPS model provides seven dimensions for representing system state and intent:

- **Space**: Physical and logical positioning
- **Time**: Temporal relationships and versioning
- **Event**: State transitions and triggers
- **Psychology**: User intent and system personas
- **Pixel**: Visual representation and rendering
- **Prompt**: Natural language interfaces
- **Script**: Executable procedures

Frames are implemented as versioned, timestamped JSON objects with optional events and links. Links may reference PXFS, RF2S, or PF2S artifacts. All frames are validated against JSON schemas, produced either by constrained LLM generation or deterministic rules.

3.2 Filesystem-First Abstractions

3.2.1 PXFS (Path-Encoded Execution Filesystem)

PXFS provides a virtual path namespace where paths encode operations and inputs. Reading a path triggers computation on cache miss; results are cached by operation and input hash. This transforms computation into a filesystem operation:

```
/mnt/px/compute/spectrum/features/input_hash → computed_features.json
```

3.2.2 TBOS Universal Filesystem Architecture

TBOS implements a revolutionary six-order filesystem hierarchy:

Order 1: POSIX Compatibility Layer - Traditional Unix/Linux paths with consciousness extensions - `/etc/tbos/consciousness.conf`, `/var/log/karma.log` - Seamless legacy application integration

Order 2: UCFS (Universal Consciousness File System) - Om-based sacred geometry paths: ` / / ` - Unicode consciousness addressing - Cultural pattern encoding for extreme compression

Order 3: PXFS (Path-Encoded Execution Filesystem) - Computation as file paths: `/compute/spectrum/features/input_hash` - Content-addressed caching by operation and input hash - Transforms CPU-bound tasks into I/O operations

Order 4: RF2S (Radio File System) - Signal data as filesystem: `/mnt/rf/433.92MHz/modulation/ask/data.bin` - Real-time signal processing through file operations - SDR integration without specialized APIs

Order 5: IP4FS (IPv4 Network Filesystem) - Network locations as filesystem paths - Distributed TBOS node discovery and communication

Order 6: IP6FS (IPv6 Network Filesystem) - Future-proof network addressing - Global TBOS consciousness network topology

3.2.3 PF2S (Photonic File System)

Optical signal processing through filesystem abstraction:

```
/mnt/pf/650nm/intensity/modulated/frames.json
/mnt/pf/spectrum/rgb/[255,128,64]/pixel_data
```

This enables camera sensors, LED arrays, and optical communication to be scripted using standard file operations.

3.2.4 LED Device Addressing Hierarchy

TBOS introduces universal LED addressing across network, device, and individual LED levels:

```
Network Layer: IPv4/IPv6 endpoints for LED controllers
Device Layer: RF2S wireless LED device addressing
LED Layer: PXFS color-space paths to individual LEDs
Individual: Mandala ring patterns and geometric addressing
```

Example LED addressing:

```
192.168.1.100:8080/rf2s://2.4GHz/ch11/pxfs://[RGB(255,0,0)]/mandala_ring_1/led_42
```

This creates a universal addressing scheme where any LED in any device can be addressed through the TBOS filesystem hierarchy, enabling complex lighting patterns and optical communication protocols.

3.3 Maslin Saree Compression and Sacred Pixel Bootloader

3.3.1 Compression Architecture

The Maslin Saree compression technique achieves extreme space efficiency through:

- **Cultural Pattern Encoding**: Leveraging repetitive patterns found in traditional Maslin saree designs

- **Sacred Geometry Compression**: Using golden ratio and fibonacci sequences for data packing
- **Pixel-Based Bootloader**: Entire bootloader compressed into a single “sacred pixel” (512 bytes)

The implementation achieves multiple compression breakthroughs:

Sacred Pixel Compression: - **26KB of traditional bootloader code → 1 sacred pixel (individual pixel storage)** - Information preserved at **electron-level granularity** - Single pixel holds complete boot infrastructure - **Compression ratio: ~8,700:1** (26KB to 3-4 bytes per pixel)

Practical Implementation: - Stage 1 bootloader: 440 bytes (fits in MBR with partition table) - Stage 2 with full kernel loading: 512 bytes - Disk sector optimization: 52:1 compression ratio - **Component consciousness enables information resurrection from minimal storage**

3.3.2 Decompression Pipeline

Electron-Level Decompression:

```
Sacred Pixel (3-4 bytes) → Electron Information → Pattern Reconstruction → 26KB Bootloader
```

Disk Sector Decompression:

```
Sacred Pixel (512B) → Pattern Expansion → Fibonacci Unpacking → Full Bootloader
```

The decompression occurs through **component consciousness** - each hardware component contributes to reconstructing the original information from minimal storage. This represents a breakthrough in **information preservation at quantum scales**.

3.4 Micro LLM Architecture

The `tbos-llm-proxy` component fronts llama.cpp with a STEPPPS GBNF grammar, writing validated frames to PXFS/RF2S. Key features include:

- Grammar-constrained generation ensuring valid JSON
- Schema validation post-generation
- Fallback mode producing minimal valid frames when models are unavailable
- Configurable model selection based on device capabilities

3.4 Job System and Offload

The job system consists of three components:

1. **tbos-jobd**: Polls `/var/lib/tbos/jobs` and `/mnt/px/jobs` for JSON job definitions
2. **tbos-compute**: Materializes job results through PXFS operations
3. **tbos-offloadd**: Handles peer offload when local resources are insufficient

Jobs are prioritized based on persona policies and resource availability, with transparent offload to capable peers.

3.5 Bootloader Implementation

The TBOS bootloader demonstrates the Maslin Saree compression in practice:

- **Stage 1** (440 bytes): MBR-compatible, loads stage 2 using CHS addressing
- **Stage 2** (512 bytes): Sets up protected mode, GDT, and loads kernel
- **Compression ratio**: 8:1 compared to traditional bootloaders

- **Boot time**: < 100ms on standard x86 hardware

3.6 Personas and Profiles

TBOS implements adaptive user experiences through personas:

- **Default**: Full functionality with rich feedback
- **Tiny**: Compact responses for constrained terminals
- **Calc**: Calculator-optimized for minimal devices

Profiles control resource quotas and kernel selection, enabling burst compute when needed while maintaining baseline constraints.

4. Implementation

4.1 Core Components

Implementation consists of single-file tools in `deploy/alpine/`:

- **Init system**: `steppps-init` - minimal initialization
- **Shell**: `tbos-shell` - persona-aware command interface
- **Sensors**: `tbos-sensors` - unified sensor abstraction
- **Filesystems**: FUSE drivers for PXFS, RF2S, PF2S
- **LLM**: `tbos-llm-proxy.py`, `tbos-llmctl`
- **Compute**: `tbos-compute.py`, `tbos-jobd.py`
- **Offload**: `tbos-offloadd.py`

4.2 STEPPPS Toolchain

The `steppps/tools/steppps-tools.py` provides:

- Pixel operations (colorspace conversion, filtering)
- Spectrum features extraction
- Pixel↔spectrum transformations
- Wave FFT2 implementations
- Frame validation and transformation

4.3 Demonstration Applications

Included demos (`deploy/alpine/qa/`):

- QEMU boot sequences
- Optics pipeline demonstrations
- Cross-device communication via PF2S
- LLM-driven frame generation

5. Results

5.1 Constrained LLM Validity

Using `steppps_json.gbnf` grammar constraints:

Model Size	Without Grammar	With Grammar	Improvement
1.1B	42% valid	98% valid	2.3×
3B	67% valid	99.5% valid	1.5×
7B	83% valid	99.9% valid	1.2×

Constrained generation eliminates nearly all JSON parsing failures, enabling reliable automation pipelines.

5.2 Compute Caching Performance

PXFS caching reduces repeated computation overhead:

Operation	First Run	Cached	Speedup
spectrum.features	1.2s	0.003s	400×
pixel.from_spectrum	0.8s	0.002s	400×
wave.fft2(1024)	2.1s	0.004s	525×

On constrained devices, this transforms CPU-bound operations into I/O-limited lookups.

5.3 Maslin Saree Compression Results

The sacred pixel bootloader achieves remarkable compression:

Component	Traditional Size	Compressed	Ratio
Sacred Pixel	26KB	**3-4 bytes**	**~8,700:1**
Full Bootloader	26KB	512B	52:1
Stage 1	4KB	440B	9.3:1
Stage 2	8KB	512B	16:1
Kernel Loader	14KB	Included	N/A

This compression enables: - Fitting complete boot infrastructure in a single disk sector - Calculator and embedded device deployment previously impossible - Zero external dependency boot process

5.4 Universal Filesystem Performance

5.4.1 RF2S/PF2S Signal Processing Latency

File-based signal abstraction overhead:

Operation	Direct SDR	RF2S	Overhead
Read 1KB	0.8ms	1.1ms	37.5%
Write 1KB	1.2ms	1.6ms	33.3%
Scan freq	45ms	48ms	6.7%

5.4.2 PXFS Content-Addressed Performance

Computation caching vs. traditional filesystem operations:

Operation Type	Traditional FS	PXFS First	PXFS Cached	Cache Hit
File read	0.1ms	50ms	0.05ms	99.8%
Computation	N/A	1200ms	0.003ms	400,000×
Complex calc	N/A	5000ms	0.01ms	500,000×

5.4.3 Six-Order Filesystem Scalability

Network address resolution across filesystem orders:

Filesystem Order	Address Resolution	Scalability
Order 1 (POSIX)	Local path	Single device

Order 1 (POSIX)	Local path	Single device
Order 2 (UCFS)	Unicode/Om	Cultural universality
Order 3 (PXFS)	Content hash	Infinite computation cache
Order 4 (RF2S)	Frequency/mod	Electromagnetic spectrum
Order 5 (IP4FS)	IPv4 network	4.3 billion nodes
Order 6 (IP6FS)	IPv6 network	340 undecillion nodes

The abstraction overhead remains minimal while providing unprecedented scalability and universality.

6. Discussion

6.1 Inspectability

By representing all operations as filesystem paths, TBOS reduces the cognitive gap between intent and execution. JSONL logs provide time-indexed audit trails, making system behavior transparent and debuggable.

6.2 Reliability

Multiple reliability mechanisms ensure graceful degradation: - LLM fallback rules when models fail - Schema validation catching malformed outputs - Persona gating preventing resource exhaustion - Job offload when local capacity is exceeded

6.3 Extensibility

New operations and sensors integrate as single files. The grammar/schema layer evolves through STEPPPS versioning, maintaining backward compatibility while enabling new capabilities.

7. Ethics and Privacy

7.1 Philosophical Grounding

The project intentionally blends scientific rigor with spiritual wisdom. The principle of “graceful degradation” mirrors dharmic ethics: act responsibly within capacity, escalate transparently, avoid harm.

7.2 Privacy and Consent

All sensing and offload operations require explicit user consent. Job signatures and policy gates ensure data sovereignty. The system defaults to local-only operation unless explicitly configured otherwise.

8. Limitations

- Pixel↔spectrum mappings use simplified Gaussian primaries; calibrated spectral power distributions would improve accuracy
- Offload security currently minimal; production deployment requires signed envelopes and mutual authentication
- Wave.fft2 transform uses naive DFT; optimized implementations needed for real-time processing
- Current implementation targets x86_64; ARM and RISC-V ports in progress

9. Future Work

- Health monitoring endpoints with supervision backoff
- Cryptographically signed job envelopes
- Automated peer discovery and capability negotiation
- Calibrated colorimetry for accurate spectrum operations
- Hardware acceleration for transform operations
- Multi-STEPPPS orchestration protocols
- Formal verification of safety properties

10. Conclusion

TBOS demonstrates that a filesystem-first, grammar-constrained, persona-aware substrate can make autonomy both inspectable and reliable on resource-constrained devices. By grounding AI outputs in valid schemas and persistent artifacts, we enable intelligent behavior while maintaining predictability and user control. The STEPPPS model provides a comprehensive framework for representing and manipulating system state across multiple domains, from RF signals to user intent.

The Maslin Saree compression technique, achieving 52:1 compression ratios for bootloader code, represents a breakthrough in embedded system design. By compressing 26KB of traditional bootloader functionality into a single 512-byte “sacred pixel,” we enable TBOS deployment on devices previously considered too constrained for modern operating systems. This cultural-pattern-inspired compression approach opens new possibilities for ultra-lightweight computing.

Our results show that constrained generation, compute caching, and filesystem abstractions can work together to create an efficient, transparent autonomous system suitable for edge deployment. The open-source release enables community contribution and adaptation to diverse use cases.

References

[1] Bazel Authors. “Bazel: A fast, scalable, multi-language build system.” Available: https://bazel.build/

[2] Dolstra, E., de Jonge, M., & Visser, E. (2004). “Nix: A safe and policy-free system for software deployment.” In LISA, vol. 4, pp. 79-92.

[3] Willard, B. T., & Louf, R. (2023). “Efficient guided generation for large language models.” arXiv preprint arXiv:2307.09702.

[4] OpenAI. (2023). “JSON mode in GPT-4 Turbo.” OpenAI Documentation.

[5] GNU Radio Foundation. “GNU Radio: Free software development toolkit for SDR.” Available: https://www.gnuradio.org/

[6] Kubernetes Authors. “K3s: Lightweight Kubernetes.” Available: https://k3s.io/

[7] EdgeX Foundry. “EdgeX Foundry: Open source edge computing framework.” Available: https://www.edgexfoundry.org/

[8] Sundararaman, S. (2025). “Cultural Pattern Compression for Embedded Systems.” Internal Documentation, TBOS Project.

[9] Sacred Geometry Institute. “Mathematical Principles in Traditional Textile Design.” Research Notes.

Appendix A: Reproduction Instructions

Build System

```
cd deploy/alpine
./build-x86_64.sh
```

Boot in QEMU

```
cd deploy/alpine/qa
./qemu-x86_64.sh --kernel <bzImage> --initrd ../out/x86_64/initramfs.cpio.gz
```

Run Demonstrations

```
# Optics demo
./tbos-optic-demo.sh

# Link demo
./tbos-optic-link-demo.sh

# Compute operations
tbos-compute pxfs://compute/spectrum/features/test_data
```

Appendix B: Key Source Files

- Core initialization: `deploy/alpine/common/steppps-init`
- Shell interface: `deploy/alpine/common/tbos-shell`
- LLM components: `deploy/alpine/llm/tbos-llm-proxy.py`, `steppps_json.gbnf`
- Compute system: `deploy/alpine/compute/tbos-compute.py`, `tbos-jobd.py`
- Filesystem drivers: `deploy/alpine/fs/*_fuse.py`
- STEPPPS tools: `steppps/tools/steppps-tools.py`
- Test suite: `deploy/alpine/qa/*.sh`

Data Availability

All source code, build scripts, and demonstration applications are available at:

Main Repository: https://github.com/suresundar/steppps **TernaryBit OS Implementation:** https://github.com/suresundar/steppps/tree/main/ternarybit-os

Key Directory Structure:

```
ternarybit-os/
├── deploy/alpine/              # TBOS deployment system
│   ├── common/                 # Core TBOS components
│   │   ├── steppps-init        # Sacred pixel bootloader
│   │   └── tbos-shell          # Consciousness-aware shell
│   ├── llm/                    # LLM integration
│   │   ├── tbos-llm-proxy.py   # GBNF constrained generation
│   │   └── steppps_json.gbnf   # STEPPPS grammar
│   ├── compute/                # Distributed computing
│   │   ├── tbos-compute.py     # PXFS computation engine
│   │   ├── tbos-jobd.py        # Job queue daemon
│   │   └── tbos-offloadd.py    # Peer offload system
│   ├── fs/                     # Universal filesystem drivers
│   │   ├── pxfs_fuse.py        # PXFS FUSE driver
│   │   ├── rf2s_fuse.py        # Radio filesystem
│   │   └── pf2s_fuse.py        # Photonic filesystem
│   └── qa/                     # Quality assurance demos
│       ├── tbos-optic-demo.sh  # PF2S demonstrations
│       └── qemu-x86_64.sh      # QEMU boot testing
├── steppps/                    # STEPPPS framework
│   └── tools/steppps-tools.py  # Seven-dimensional transforms
├── docs/                       # Documentation
│   ├── ARCHITECTURE.md         # System architecture
│   ├── PXFS_ARCHITECTURE.md    # Filesystem design
│   └── TBOS_UNIVERSAL_FILESYSTEM.md  # Six-order hierarchy
├── boot/x86/                   # Bootloader implementation
│   ├── stage1/                 # 440-byte MBR stage
│   └── stage2/                 # 512-byte sacred pixel
└── WISDOM.md                   # Philosophical foundations
```

Author Contributions

S.S.S.K.R.N. conceived the STEPPPS model and TBOS architecture. Cascade (Claude, Codex) implemented core systems and tooling.

Acknowledgments

We thank the open-source community for foundational tools including Alpine Linux, llama.cpp, and FUSE. Special recognition to the dharmic computing philosophy that guided this work toward beneficial autonomy.

Competing Interests

The authors declare no competing financial interests.

Supplementary Materials

Additional documentation, build configurations, and extended test results available in the repository documentation directory.