

Feature: Memory-Aware File Compression Optimizer

Core Idea

Instead of:

“Compress all JPEG files”

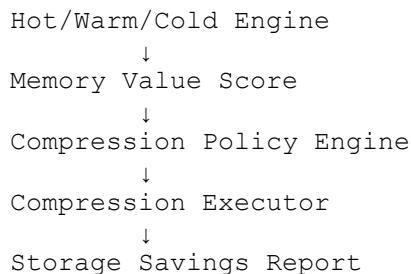
You do:

“Compress files based on how important they are.”

So compression is based on **memory value**, not file type.



High-Level Architecture



Complete Tech Stack

Since you're already building previous modules:

Backend

- FastAPI
- Python

Database

- SQLite

Compression Tools

File Type	Library
Generic	zipfile
Images	Pillow
PDF	PyMuPDF / pikepdf
Videos	ffmpeg (optional demo)

For hackathon → zipfile + Pillow is enough.



Step 1: Use Memory Engine Output

From your Hot/Warm/Cold system, each file has:

```
heat_score
memory_type (Hot / Warm / Cold)
importance_score
```



Step 2: Define Compression Policies

Create rule-based policies.

Policy Table

Memory Type	Compression Level	Strategy
🔥 Hot	No compression	Keep fast access
☁️ Warm	Mild compression	ZIP standard
⌘ Cold	Aggressive compression	High ratio



Step 3: Compression Strategy Design

We create adaptive logic.



HOT Files

- No compression
- Or very light zip
- Keep original format

```
if memory_type == "Hot":
```

```
return "No Compression Recommended"
```

WARM Files

- ZIP compression
- Medium compression level

```
import zipfile

def compress_warm(file_path):
    with zipfile.ZipFile(file_path + ".zip",
                          "w",
                          compression=zipfile.ZIP_DEFLATED,
                          compresslevel=5) as zipf:
        zipf.write(file_path)
```

COLD Files

Aggressive compression.

If Image → Reduce quality

```
from PIL import Image

def compress_image_aggressive(path):
    img = Image.open(path)
    img.save("compressed.jpg", quality=30, optimize=True)
```

If General File → High ZIP compression

```
compresslevel=9
```



Step 4: Adaptive Compression Logic

Full logic:

```
def compression_engine(file):
    if file.memory_type == "Hot":
        return "Skip"

    elif file.memory_type == "Warm":
        compress_zip(file.path, level=5)

    elif file.memory_type == "Cold":
        if file.extension in ["jpg", "png"]:
            compress_image_aggressive(file.path)
        else:
            compress_zip(file.path, level=9)
```

Step 5: Estimate Storage Savings

Before compression:

```
original_size
```

After compression:

```
compressed_size
```

Calculate:

```
space_saved = original_size - compressed_size
```

Display:

“2.4 GB saved by compressing 34 cold memories.”

This is powerful in demo.



Dashboard Design

Show:

Memory Type Total Size Potential Savings

Hot	5GB	0
Warm	12GB	1.2GB
Cold	24GB	8.7GB

Add button:

```
Optimize Storage
```

Intelligent Enhancement (Judges Will Love This)

Add “Risk Score”

If file:

- Recently accessed → low compression

- Linked to other files → low compression

Add safety rule:

```
if open_count > 20:  
    avoid aggressive compression
```



Advanced (Optional)

Use clustering to detect:

- Screenshots
- Repeated images
- Duplicate projects

Then compress batch-wise.



Folder Structure

```
compression-engine/  
    └── backend/  
        ├── compression_engine.py  
        ├── policy_manager.py  
        ├── heat_engine.py  
        └── storage_analyzer.py  
  
    └── frontend/  
        └── OptimizationDashboard.jsx  
  
    └── reports/
```



Complete Execution Plan

Phase 1 – Basic Working Demo

- Connect memory engine
 - Create rule-based compression
 - Show storage saved
-

Phase 2 – Smart Policies

- Adaptive compression per type
 - Safety rules
-

Phase 3 – WOW Layer

- Predict future storage exhaustion
- Show “Storage Health Score”

Example:

“Your storage health is 63%. Optimizing cold memories can improve it to 82%.”

That is hackathon gold.

⚠ Important Practical Advice

For hackathon:

DO NOT compress real system files.

Instead:

- Create demo dataset
- Simulate compression
- Show before/after size

Focus on concept + visualization.



Why This Is Strong for SanDisk

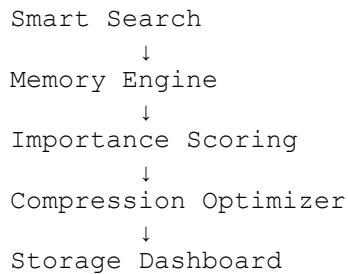
Because SanDisk cares about:

- Storage efficiency
- Performance tiers (SSD vs HDD)
- Data lifecycle

You are literally building:

AI-powered data lifecycle manager

Final Architecture (All Features Combined)



Now your product becomes:

AI Personal Memory Assistant

Final Recommended Stack For YOU

Since you're already handling ML + backend:

```
FastAPI  
SQLite  
SentenceTransformers  
FAISS  
Pillow  
zipfile  
React + Chart.js
```

Keep everything modular.

Killer Presentation Line

Say this:

“Traditional compression tools compress based on file type. Our system compresses based on memory value.”

That's innovation.
