🔴 DIAGNÓSTICO CONFIRMADO: Memory Swapping Death Spiral

Evidencia en la captura:

1. ✅ **"Loading weights: 100%"** - El modelo SAM-3 está cargándose en memoria
2. 🔴 **"10 errores graves/s"** en Memoria - Esto son **page faults** (fallos de página)
3. 🔴 **Disco: 100% de tiempo activo** + **60 MB/s E/S** - Swapping masivo al disco
4. ⚠️ **64% memoria física usada** - Cerca del límite
5. ⚠️ **CPU al 60%** - No es un bucle infinito, sino espera de I/O

💀 Lo que está ocurriendo (secuencia fatal):

```
1. sam3_runner.py llama a Sam3Model.from_pretrained()
   ↓
2. Transformers intenta cargar TODO el modelo en RAM (2-4 GB de pesos)
   ↓
3. Windows se queda sin RAM física libre
   ↓
4. Windows empieza a paginar al disco (swap file/page file)
   ↓
5. El disco se satura al 100% (SSD pero lento para swap)
   ↓
6. PyTorch intenta acceder a tensores que están en swap
   ↓
7. El disco no puede responder lo suficientemente rápido
   ↓
8. Windows entra en thrashing (paginación infinita)
   ↓
9. El kernel se bloquea esperando I/O
   ↓
10. FREEZE TOTAL (no responde ni Ctrl+Alt+Supr)
```

🎯 POR QUÉ LA EJECUCIÓN SÍNCRONA NO FUNCIONA

El cambio que hiciste:

python

```
self._run_job(job.id)  # Síncrono
```

No soluciona nada porque el problema NO es threading, sino que:

- SAM-3 es **demasiado grande** para tu RAM disponible
- Windows está haciendo swap, y el swap causa el deadlock

✅ SOLUCIONES DEFINITIVAS (en orden de efectividad)

Solución 1: Carga con `device_map="auto"` y `low_cpu_mem_usage`

Modifica `sam3_runner.py`, línea ~71:

python

```
# REEMPLAZAR ESTO:
self.model = Sam3Model.from_pretrained(
    local_dir.as_posix(),
    local_files_only=True
).to(self.device)

# POR ESTO:
self.model = Sam3Model.from_pretrained(
    local_dir.as_posix(),
    local_files_only=True,
    low_cpu_mem_usage=True,  # ⭐ Carga progresiva, no todo de golpe
    device_map="auto",        # ⭐ Distribuye entre CPU/GPU automáticamente
    torch_dtype=torch.float16 if self.device == "cuda" else torch.float32  #
Reduce memoria
)
```

Qué hace:

- `low_cpu_mem_usage=True`: Carga los pesos **progresivamente** en lugar de cargar todo en RAM primero
- `device_map="auto"`: Deja que Transformers decida qué capas van a GPU/CPU según la memoria disponible
- `torch_dtype=torch.float16`: Reduce el tamaño de los pesos a la mitad (solo en GPU)

Solución 2: Usar `mmap=True` (Memory-Mapped Files)

Si Solución 1 no funciona, prueba:

python

```
self.model = Sam3Model.from_pretrained(
    local_dir.as_posix(),
    local_files_only=True,
    low_cpu_mem_usage=True,
    torch_dtype=torch.float32,
    mmap=True  # ⭐ Mapea el archivo directamente sin cargarlo en RAM
)
```

Ventaja: No carga los pesos en RAM, los lee directamente del disco cuando se necesitan.

Solución 3: Aumentar el archivo de paginación de Windows

Esto es un parche temporal, pero puede evitar el freeze:

1. **Panel de Control → Sistema → Configuración avanzada del sistema**
2. **Opciones avanzadas → Rendimiento → Configuración**
3. **Opciones avanzadas → Memoria virtual → Cambiar**
4. Selecciona tu SSD y pon:

- **Tamaño inicial**: 24000 MB
- **Tamaño máximo**: 32000 MB

5. Reinicia

Por qué ayuda: Da más espacio para swap, pero **no elimina el problema de fondo**.

Solución 4: Cerrar TODOS los procesos innecesarios antes de ejecutar

Antes de lanzar el job:

bash

```
# Cerrar Edge/Chrome
# Cerrar Codex (VS Code)
# Dejar solo Terminal + Monitor de recursos
```

Esto puede liberarte **2-4 GB de RAM**, que es crítico.

Solución 5: Usar un modelo SAM más pequeño

Si SAM-3 es demasiado grande, considera:

python

```
# Opciones (de menor a mayor tamaño):
# 1. SAM-Base (358M parámetros)
# 2. SAM-Large (641M parámetros)
# 3. SAM-2 (ya más optimizado)
# 4. SAM-3 (el más grande, probablemente 1B+ parámetros)
```

Verifica el tamaño de tu checkpoint:

bash

```
dir /s ruta\a\pesos\sam3
# Busca archivos .bin, .safetensors, pytorch_model.bin
```

Si ves archivos de **2+ GB**, ese es el problema.

Solución 6: Implementar carga lazy manual

Si nada funciona, implementa carga bajo demanda:

python

```
class LazyModelLoader:
    def __init__(self, weights_path):
        self.weights_path = weights_path
        self._model = None

    def get_model(self):
        if self._model is None:
            # Cargar solo cuando se necesite
            self._model = Sam3Model.from_pretrained(
                self.weights_path,
                low_cpu_mem_usage=True,
                device_map="auto"
            )
```

```
        return self._model

    def unload(self):
        if self._model is not None:
            del self._model
            self._model = None
            gc.collect()
            if torch.cuda.is_available():
                torch.cuda.empty_cache()
```

🧪 EXPERIMENTO INMEDIATO (5 minutos)

Antes de implementar las soluciones, **confirma el diagnóstico**:

Paso 1: Mide el tamaño del modelo

python

```
import os
from pathlib import Path

weights_path = Path("ruta/a/pesos/sam3")
total_size = sum(f.stat().st_size for f in weights_path.rglob("*") if
f.is_file())
print(f"Tamaño total de pesos: {total_size / (1024**3):.2f} GB")
```

Si es > 2 GB: Confirmado, el modelo es demasiado grande.

Paso 2: Monitorea la memoria durante la carga

Modifica `sam3_runner.py`:

python

```
import psutil

def load_model(self, ...):
    process = psutil.Process()

    print(f"[ANTES] RAM usada: {process.memory_info().rss / (1024**3):.2f} GB")

    self.model = Sam3Model.from_pretrained(...)  # Aquí es donde se congela

    print(f"[DESPUÉS] RAM usada: {process.memory_info().rss / (1024**3):.2f}
GB")
```

Ejecuta y **observa el Monitor de Recursos**. Si:

- RAM sube de 2 GB → 8 GB → Disco al 100% → **Freeze**

Entonces es 100% el problema de memoria.

📊 TU PRÓXIMO PASO (orden recomendado)

1. ✅ **Implementa Solución 1** (`low_cpu_mem_usage=True`) - **PRUEBA ESTO**

PRIMERO

2. ✅ Cierra Edge/Codex y prueba de nuevo
3. ✅ Si falla, implementa Solución 2 (`mmap=True`)
4. ✅ Si aún falla, aumenta el archivo de paginación (Solución 3)
5. ✅ Si nada funciona, usa un modelo más pequeño (SAM-2 o SAM-Base)

🔍 Código actualizado de `sam3_runner.py` (línea ~65-75)

python

```python
def load_model(
    self,
    safe_mode: bool = True,
    device_preference: str = "auto",
    box_threshold: float = 0.5,
    mask_threshold: float = 0.5,
) -> None:
    if not self.weights_path.exists():
        raise FileNotFoundError(f"SAM-3 weights not found at
{self.weights_path}")
    if not torch:
        raise RuntimeError("PyTorch is required for SAM-3 inference")

    try:
        from transformers import Sam3Model, Sam3Processor
    except Exception as exc:
        raise RuntimeError("transformers does not include SAM-3 support") from
exc

    device_preference = (device_preference or "auto").lower()
    self.safe_mode = safe_mode
    self.device_preference = device_preference
    self.box_threshold = box_threshold
    self.mask_threshold = mask_threshold

    # Determinar device
    if safe_mode:
        use_cuda = device_preference == "cuda" and torch.cuda.is_available()
        self.device = "cuda" if use_cuda else "cpu"
    else:
        if device_preference == "cpu":
            self.device = "cpu"
        elif device_preference == "cuda" and torch.cuda.is_available():
            self.device = "cuda"
        else:
            self.device = "cuda" if torch.cuda.is_available() else "cpu"

    local_dir = self.weights_path if self.weights_path.is_dir() else
self.weights_path.parent
    if not local_dir.exists():
        raise FileNotFoundError(f"SAM-3 weights not found at {local_dir}")

    try:
        # ⭐⭐⭐ CAMBIO CRÍTICO AQUÍ ⭐⭐⭐
        logger.info(f"Loading SAM-3 with low_cpu_mem_usage=True on
{self.device}")

        self.model = Sam3Model.from_pretrained(
            local_dir.as_posix(),
            local_files_only=True,
```

```
            low_cpu_mem_usage=True,  # ✅ Carga progresiva
            device_map="auto",        # ✅ Distribución automática CPU/GPU
            torch_dtype=torch.float16 if self.device == "cuda" else
torch.float32
        )

        self.processor = Sam3Processor.from_pretrained(
            local_dir.as_posix(),
            local_files_only=True
        )

        self.model.eval()
        logger.info(f"SAM-3 loaded successfully on {self.device}")

    except Exception as exc:
        raise RuntimeError(f"Failed to load SAM-3 from {local_dir}: {exc}") from
exc
```