DeDupe — Informe para Codex

Autor: Víctor (por medio de ChatGPT)
Ámbito: Revisión técnica de GUI/CLI y módulos `DeDupe.*`
Objetivo: Dejar una guía clara y accionable para (1) reproducir el problema, (2) validar el estado de los módulos y (3) proponer un plan de corrección seguro.

---

1) Resumen ejecutivo

- La **GUI actual** se congela al pulsar *Simulación/Ejecutar*. No hay progreso visible ni actividad real en destino.
- La **CLI** permanece abierta, pero emite sólo una secuencia numérica sin relación con el pipeline.
- El **código de módulos** en `Modules.zip` muestra una arquitectura sólida (pipeline → hashing → grouping → dedupe → quarantine → summary), pero la **orquestación UI** está acoplada al hilo de WPF y eso explica el congelamiento.
- Propuesta: **desacoplar ejecución** (ThreadJob o proceso CLI no interactivo) y **streaming de progreso** por archivo `.progress` + *tail* en la GUI.
- Diseño: suavizar tema oscuro (gris #202020/#252525 + texto #E0E0E0) para mejorar legibilidad.

---

2) Cómo reproducir (estado observado)

1. Arrancar `DeDupe — GUI` sin tocar parámetros.
2. Marcar **Exportar resumen JSON** y pulsar **Ejecutar**.
3. Resultado: **ventana congelada**; la CLI no muestra progreso y no se crean artefactos en Cuarentena/Logs/Out.
4. El disco muestra actividad intermitente, pero no hay efectos.

   Nota: la combinación blanco/negro en la GUI produce contraste agresivo; se sugiere paleta gris oscuro.

---

3) Arquitectura actual (a alto nivel)

- **Pipeline** (`DeDuPe.Pipeline`): expone `Invoke-DeDupePipeline` con `-OnTick`, `-ReportIntervalMs`, `-Run`, etc.; inicia **ticker de progreso** desde `DeDuPe.Metrics.Ultra`.
- **Métrica** (`DeDuPe.Metrics.Ultra`): capa de alto rendimiento (ETA/MBps/EPS, `Start-ProgressTicker` / `Stop-ProgressTicker`, tuning de ThreadPool, writer JSONL rápido).
- **Logging** (`DeDuPe.Logging`): versión actual implementa logger propio (crea archivo, `Add-Content` + **rotación** por tamaño, `ConvertTo-PlainObject` para serializar). Hay un `*.orig` que delegaba en `Metrics.Ultra`.
- **Hashing** (`DeDuPe.Engine` + `DeDuPe.Hashing.MetricsAdapter`): C# hot-path con progresos (`IProgress<long>`) y versión PS robusta con *streaming* por bloques.

- **Grouping** y **DedupeByHash**: agrupan por tamaño/hash y deciden *survivors* según estrategia (Oldest/Newest/ShortestPath).
- **Quarantine**: mueve duplicados restantes (layout `Flat` / `BySize`).

Conclusión: los módulos de negocio están OK; el problema está en **cómo la GUI orquesta** el pipeline (bloquea el hilo UI).

---

4) Hallazgos en los módulos (lectura rápida del ZIP)

4.1 `DeDuPe.Pipeline.psm1`

- Exporta `Invoke-DeDupePipeline` con `-OnTick` y gestiona progreso vía `DeDuPe.Metrics.Ultra\Start-ProgressTicker`.
- Incluye enumeración, hashing paralelo (`Invoke-HashRecordsParallel`), *grouping*, dedupe y quarantine, con **telemetría** a JSONL.
- Observación: existe `DeDuPe.Pipeline.psm1.copy` (versión previa). Cambios netos (aprox.): **+45 / −146** líneas vs. actual → hubo refactor.
- Recomendación: mantener la **firma pública** estable (especialmente `-OnTick`, `-ReportIntervalMs`, `-Run`) para que la GUI/CLI no rompan compatibilidad.

4.2 `DeDuPe.Metrics.Ultra.psm1`

- Define `Start-ProgressTicker` (Timer .NET) y `Stop-ProgressTicker`.
- Métricas por EMA: MB/s y EPS estables + ETA.
- Utilidades de rendimiento: ThreadPool tuning, canales acotados, writer JSONL, `Copy-ItemWithETA`.
- **Bien**: el ticker corre fuera del hilo UI; es seguro para backend.

4.3 `DeDuPe.Logging.psm1` **(actual) vs** `DeDuPe.Logging.psm1.orig`

- **Actual**: logger propio → crea/rota archivo y serializa con `ConvertTo-PlainObject`.
- **Orig**: delgado, delegaba todo a `Metrics.Ultra` (writer de alto rendimiento).
- Riesgo menor: múltiples escritores concurrentes podrían bloquear si en el futuro hay varios procesos; hoy el pipeline escribe desde un solo contexto, por lo que **no es bloqueo**.

4.4 `DeDuPe.Hashing.MetricsAdapter.psm1`

- Hash incremental por bloques con reporte opcional a `Update-HiResMetrics`.
- Correcto para escenarios sin `DeDuDuEngine` (C#) o como *fallback*.

4.5 `DeDuPe.Engine.psm1`

- C# hot-path (SHA-256 + mini-hash + compare byte-a-byte + hardlinks).
- Uso de `FileStream` con **SequentialScan** y buffer grande: ✔️
- Callbacks de progreso (`IProgress<long>`) compatibles con la capa de métricas.

Diagnóstico principal: No hay defectos obvios en estos módulos para explicar el *freeze*; el problema está en la **invocación desde la GUI** (orquestación síncrona en el hilo STA + tail/logs/timers que dependen del Dispatcher).

---

5) Causa probable del congelamiento en GUI

1) `Invoke-DeDupePipeline` se ejecuta **en el mismo hilo** de WPF; mientras corre, el Dispatcher no procesa mensajes.
2) El `-OnTick` intenta actualizar progreso (directa o indirectamente) y queda **encolado** hasta que el hilo UI se libere.
3) Los *tailers* (JSONL y progreso) existen, pero si el hilo UI no respira, el usuario **no ve nada**.

---

6) Plan de corrección (seguro y reversible)

Fase 1 — *Hotfix* (1 PR)

- [ ] Ejecutar el pipeline en **ThreadJob** (o proceso `pwsh` dedicado) → la GUI **nunca** toca el backend en el hilo UI.
- [ ] `-OnTick` del backend escribe snapshots en `actions.progress` (JSONL, una línea/snapshot).
- [ ] GUI usa `DispatcherTimer` (250–500 ms) para *tailear* `actions.progress` y actualizar **barra/ETA/MB/s**.
- [ ] Tail del **log JSONL** permanece (ya implementado) pero con `FileShare.ReadWrite`.
- [ ] Botones **deshabilitados** durante la corrida; se re-habilitan al completar o fallar.
- [ ] Manejo de errores: `MessageBox` con `Exception.ToString()` (para no perder `InnerException`).

Fase 2 — Opcional (arquitectura CLI-first)

- [ ] Añadir **modo** `-NonInteractive` a `DeDupe.ps1` (CLI) para aceptar parámetros.
- [ ] La GUI hace de **front-end**: construye argumentos y lanza `DeDupe.ps1 -NonInteractive ...`.
- [ ] Reutiliza los mismos timers de *tail* (log y `.progress`).
- [ ] Beneficio: testear backend sin GUI (CI), y la GUI no requiere mantener runspaces.

---

7) Cambios sugeridos en la GUI (extractos)

> Nota: estos fragmentos son **ilustrativos**; el PR incluirá el código completo consolidado.

```
# 1) Timer de progreso (archivo .progress)
$progressFile = "$($lpn.Text.Text).progress"
$lastProgSize = 0L
$progTimer = New-Object System.Windows.Threading.DispatcherTimer
$progTimer.Interval = [TimeSpan]::FromMilliseconds(250)
```

```
$progTimer.Add_Tick({
  try {
    if (-not (Test-Path -LiteralPath $progressFile)) { return }
    $fi = [IO.FileInfo]::new($progressFile)
    if ($fi.Length -le $lastProgSize) { return }
    $fs = [IO.File]::Open($progressFile,'Open','Read','ReadWrite')
    try {
      $fs.Position = $lastProgSize
      $sr = New-Object IO.StreamReader($fs)
      while(-not $sr.EndOfStream){
        $snap = ($sr.ReadLine() | ConvertFrom-Json)
        Update-UIProgress $snap
      }
    } finally { $lastProgSize = $fi.Length; $fs.Dispose() }
  } catch {}
})
```

```
# 2) Lanzar pipeline en runspace separado (ThreadJob)
Import-Module ThreadJob -ErrorAction SilentlyContinue | Out-Null
$btnDry.IsEnabled=$false; $btnRun.IsEnabled=$false
$progTimer.Start()
$job = Start-ThreadJob -Name 'DeDupe.Run' -ArgumentList @{
  Path=$src.Text.Text; Recurse=[bool]$chkRecurse.IsChecked; IncludeHidden=[bool]
$chkHidden.IsChecked;
  AllowZeroByte=$false; Keep=[string]$cmbKeep.SelectedItem;
QuarantinePath=$qpn.Text.Text;
  LogPath=$lpn.Text.Text; QuarantineLayout=[string]$cmbLayout.SelectedItem;
  DegreeOfParallelism=[int]$txtDop.Text; BlockSizeKB=[int]$txtBlk.Text;
Verify=[bool]$chkVerify.IsChecked;
  Apply=$apply; ProgressPath=$progressFile
} -ScriptBlock {
  param($a)
  $ErrorActionPreference='Stop'
  foreach($m in
@('DeDuPe.Metrics.Ultra','DeDuPe.Logging','DeDuPe.Grouping','DeDuPe.DedupeByHash','DeDuPe.Quarant
{
    Import-Module (Join-Path $using:modDir ("{0}.psd1" -f $m)) -Force
  }
  $tick = { param($s) try { ($s|ConvertTo-Json -Depth 6)+[Environment]::NewLine
| Add-Content -LiteralPath $using:a.ProgressPath -Encoding UTF8 } catch {} }
  Invoke-DeDupePipeline @{
    Path=$a.Path; Recurse=$a.Recurse; IncludeHidden=$a.IncludeHidden;
AllowZeroByte=$a.AllowZeroByte;
    Keep=$a.Keep; QuarantinePath=$a.QuarantinePath; LogPath=$a.LogPath;
QuarantineLayout=$a.QuarantineLayout;
    DegreeOfParallelism=$a.DegreeOfParallelism; BlockSizeKB=$a.BlockSizeKB;
```

```
ReportIntervalMs=500; Verify=$a.Verify; Run=$a.Apply; OnTick=$tick;
Confirm=$false
  }
}
Register-ObjectEvent -InputObject $job -EventName StateChanged -Action {
  if ($eventArgs.JobStateInfo.State -in @('Completed','Failed','Stopped')) {
    $using:progTimer.Stop(); $using:btnDry.IsEnabled=$true;
$using:btnRun.IsEnabled=$true
  }
} | Out-Null
```

---

8) UI/UX (rápidas)

- Paleta: fondo `#202020` / paneles `#252525` / acento `#3A96DD` / texto `#E0E0E0`.
- Dejar **visor vacío** al inicio; sólo rellenar tras primera línea del log/progreso.
- Añadir botón **Cancelar** (futuro): `Stop-ThreadJob` + soporte de cancelación en pipeline.

---

9) Checklist de validación

- [ ] La ventana **no se congela** al ejecutar.
- [ ] `actions.progress` crece con snapshots JSON.
- [ ] Barra/ETA/MBps se actualizan en tiempo real.
- [ ] Exportar resumen JSON crea archivo esperado.
- [ ] Sin errores de *Runspace* ni *InvalidOperation* de Dispatcher.

---

10) Siguiente iteración (lo que falta revisar)

- Revisión línea por línea de `DeDuPe.Pipeline.psm1` vs `*.copy` para documentar todos los cambios de lógica.
- Validar `DeDuPe.Logging` bajo estrés (rotación concurrente).
- Confirmar que `Engine` cierre **siempre** streams y buffers (ya se ve correcto, pero corremos pruebas).
- Decidir si migrar a arquitectura **CLI-first** (GUI como front-end).

Puedes comentar en este Canvas y ajusto el PR en caliente. En la siguiente iteración documento las diferencias exactas por módulo y adjunto el parche completo de la GUI.