PowerShell-based Codex CLI Pattern (Transcript Excerpt)

This PDF contains only the requested sketch from the chat.

Below is a **PowerShell “Codex CLI pattern”** you can adapt. I’m giving you **two common shapes**:

1) **Use the official Codex CLI from PowerShell** (simplest; it *can* read/change/run code in a directory)
2) **Roll your own PowerShell wrapper that sends selected files to the OpenAI Responses API** (maximum control)

Pattern A — Drive the official Codex CLI from PowerShell

Codex CLI is a local terminal agent that operates in your chosen directory.

```
# 0) Install once (example uses npm global install)
# npm i -g @openai/codex

# 1) Authenticate (one-time per machine/user)
# $env:OPENAI_API_KEY = "sk-..."  # keep in a secure store in real life
# $env:OPENAI_API_KEY | codex login --with-api-key

# 2) Run codex in a repo directory, giving it a task
Push-Location "C:\github\JimFawcett\NewSite"

# Optional safety checkpoint before letting an agent edit files
git status
git checkpoint -m "before codex change" 2>$null  # if you have a checkpoint alias; otherwise use c

# Run Codex interactively in this directory
codex

Pop-Location
```

When you want it non-interactive, you typically script: “prepare branch/checkpoint → run codex with task → review diff → commit”.

Pattern B — PowerShell wrapper that calls the OpenAI Responses API (DIY “Codex CLI”)

This pattern: **select files → build a prompt → call `POST /v1/responses` → save output (patch/diff) → optionally apply**.

1) Core script skeleton

```
param(
  [Parameter(Mandatory=$true)] [string] $RepoRoot,
  [Parameter(Mandatory=$true)] [string] $Task,
  [string] $Model = "gpt-5.2-codex",
```

```
  [int]    $MaxFileBytes = 120000,
  [string[]] $Include = @("*.rs","*.ps1","*.psm1","*.cs","*.cpp","*.h","*.hpp","*.js","*.ts","*.ht
  [string[]] $ExcludeDirs = @(".git","target","bin","obj","node_modules",".vscode"),
  [switch] $EmitPatchOnly
)

# --- 0) Preconditions ---
if (-not $env:OPENAI_API_KEY) { throw "Set OPENAI_API_KEY in your environment." }

$Endpoint = "https://api.openai.com/v1/responses"

# --- 1) Gather files (lightweight, configurable) ---
function Get-RepoFiles {
  param([string]$Root)

  $files = Get-ChildItem -Path $Root -Recurse -File -Force |
    Where-Object {
      # Exclude directories anywhere in the path
      foreach ($d in $ExcludeDirs) { if ($_.FullName -like "*\${d}\*") { return $false } }
      return $true
    } |
    Where-Object {
      # Include patterns
      foreach ($pat in $Include) { if ($_.Name -like $pat) { return $true } }
      return $false
    } |
    Where-Object { $_.Length -le $MaxFileBytes }

  return $files
}

# --- 2) Build a single prompt payload (simple + auditable) ---
function New-RepoContext {
  param([System.IO.FileInfo[]]$Files, [string]$Root)

  $sb = New-Object System.Text.StringBuilder

  [void]$sb.AppendLine("You are a coding assistant. Follow instructions exactly.")
  [void]$sb.AppendLine("Task:")
  [void]$sb.AppendLine($Task)
  [void]$sb.AppendLine("")
  [void]$sb.AppendLine("Constraints:")
  [void]$sb.AppendLine("- If you propose code changes, output a unified diff (git patch) against t
  [void]$sb.AppendLine("- Do not invent files not shown; if needed, say what additional file is re
  [void]$sb.AppendLine("")
  [void]$sb.AppendLine("Repository snapshot (file contents follow).")
  [void]$sb.AppendLine("-----")

  foreach ($f in $Files) {
    $rel = $f.FullName.Substring($Root.TrimEnd('\').Length).TrimStart('\')
    $text = Get-Content -LiteralPath $f.FullName -Raw

    [void]$sb.AppendLine("FILE: $rel")
    [void]$sb.AppendLine("

")
[void]$sb.AppendLine($text)
```

[void]$sb.AppendLine("

```
")
    [void]$sb.AppendLine("-----")
  }

  return $sb.ToString()
}

# --- 3) Call the Responses API ---
function Invoke-Codex {
  param([string]$Prompt)

  $headers = @{
    "Authorization" = "Bearer $($env:OPENAI_API_KEY)"
    "Content-Type"  = "application/json"
  }

  $body = @{
    model = $Model
    input = $Prompt
  } | ConvertTo-Json -Depth 20

  $resp = Invoke-RestMethod -Method Post -Uri $Endpoint -Headers $headers -Body $body
  return $resp
}

# --- 4) Extract text output from Responses API result ---
function Get-ResponseText {
  param($Response)

  $chunks = @()

  foreach ($o in $Response.output) {
    if ($o.type -eq "output_text" -and $o.text) { $chunks += $o.text }
    if ($o.content) {
      foreach ($c in $o.content) { if ($c.type -eq "output_text" -and $c.text) { $chunks += $c.tex
    }
  }

  return ($chunks -join "`n")
}

# --- 5) Main ---
$repo = (Resolve-Path $RepoRoot).Path
$files = Get-RepoFiles -Root $repo
if (-not $files -or $files.Count -eq 0) { throw "No files matched include/exclude filters." }

$prompt = New-RepoContext -Files $files -Root $repo
$raw = Invoke-Codex -Prompt $prompt
$text = Get-ResponseText -Response $raw

# Save output
$outDir = Join-Path $repo ".codex"
New-Item -ItemType Directory -Force -Path $outDir | Out-Null
$stamp = Get-Date -Format "yyyyMMdd-HHmmss"
```

```
$outPath = Join-Path $outDir "codex-$stamp.txt"
Set-Content -LiteralPath $outPath -Value $text -Encoding UTF8

Write-Host "Saved: $outPath"

if ($EmitPatchOnly) {
  # If you instructed the model to output a unified diff, you can try applying it:
  $patchPath = Join-Path $outDir "codex-$stamp.patch"
  Set-Content -LiteralPath $patchPath -Value $text -Encoding UTF8
  Write-Host "Patch: $patchPath"
  # Optional: apply patch (review first!)
  # git -C $repo apply --check $patchPath
  # git -C $repo apply $patchPath
}
```

2) Example invocation

```
$env:OPENAI_API_KEY = "<your key>"

.\Invoke-Codex.ps1 `
  -RepoRoot "C:\github\JimFawcett\NewSite" `
  -Task "Refactor the JS that builds the Explorer menus to reduce duplication. Keep behavior ident
  -EmitPatchOnly
```

Notes that make this pattern work well

- **Start small**: restrict to the handful of files you care about (or add a `-Paths` parameter).
- **Force “diff only”** in your prompt if you want machine-applicable changes.
- **Checkpoint with git** before applying patches.