Perfect—let’s turn everything we’ve planned into a **bullet‑proof, execution‑ready plan formatted for the “Plan” feature in Continue (VS Code)**. It’s broken into small, reviewable tasks with clear dependencies, file targets, prompts for the AI, and acceptance checks. Paste this into a PLAN.md at your repo root (or into Continue’s Plan input) and work top‑to‑bottom.

**MagicOptimizer — Continue Plan (UE 5.6, C++ + Python)**

**Goal:** Deliverable‑first, **CSV‑driven** Audit → Recommend → Apply → Verify across all pipelines (PC/Console/Mobile/VR/VP).  
**Style:** Macro → micro; ship in thin, testable slices; everything traceable to CSV + JSON in Saved/Optimizor/History/<RunId>/.

**Project root for plugin code:**  
HostProject/Plugins/MagicOptimizer/

**Pre‑flight (once)**

* Ensure UE 5.6 install is verified; .NET 8 Desktop Runtime installed.
* VS2022 workloads: Game Dev with C++, Desktop C++, v143 toolset, Win 10/11 SDK.
* UE **PythonScriptPlugin** enabled in the HostProject.
* If Marketplace plugins spam warnings (e.g., GraphNUnrealPlugin), disable for now.

**Milestone 0 — Repo Hygiene & Scaffolding**

**T0.1 Create PLAN + folders (no code changes)**

**Files/Dirs**

HostProject/Plugins/MagicOptimizer/

Content/Python/magic\_optimizer/{\_\_init\_\_.py, entry.py, io\_csv.py, utils.py}

Content/Python/magic\_optimizer/textures/{audit.py,recommend.py,apply.py,verify.py}

Content/Python/magic\_optimizer/meshes/{audit.py,recommend.py,apply.py,verify.py}

Content/Python/magic\_optimizer/materials/{audit.py,recommend.py,apply.py,verify.py}

Content/Python/magic\_optimizer/runtime/{audit.py,recommend.py,apply.py,verify.py}

Content/Python/magic\_optimizer/world/{audit.py,recommend.py,apply.py,verify.py}

Content/Python/magic\_optimizer/presets/{Mobile\_AR.json,PC\_High.json,Console.json,VR.json,VirtualProduction.json,Authoring\_Safe.json}

**Acceptance**

* Folders exist; repo builds unchanged.

**Continue prompt (paste into Plan step)**

Create empty Python package skeleton under Content/Python/magic\_optimizer with the file tree above. Use empty modules with docstrings only.

**Milestone 1 — Foundations (C++)**

**T1.1 Settings & Config structs**

**Why:** Single source of truth for runs; feeds Python.

**C++ Targets**

* Source/MagicOptimizer/Public/OptimizerSettings.h/.cpp
* Source/MagicOptimizer/Public/OptimizerRun.h/.cpp
* Source/MagicOptimizer/Public/PythonBridge.h/.cpp
* Source/MagicOptimizer/Public/SOptimizerPanel.h/.cpp
* Source/MagicOptimizer/Public/OptimizerJson.h/.cpp (optional helper)

**What to implement**

* UOptimizerSettings (EditorPerProjectUserSettings): deliverable preset, category mask, scope (selection/include/exclude), safety (dry\_run, max\_changes…), gates (apply\_only…).
* FOptimizerConfig (USTRUCT) → JSON (snake\_case).
* NewRunId(), EnsureRunDir(), SaveConfigJson().

**Acceptance**

* Editor compiles; NewRunId() creates Saved/Optimizor/History/<RunId>/config.json.

**Continue prompt**

Implement UOptimizerSettings and FOptimizerConfig for MagicOptimizer. Add JSON save/load (snake\_case) using FJsonObjectConverter. Create NewRunId() and EnsureRunDir() helpers that write to Saved/Optimizor/History/<RunId>/.

**T1.2 Nomad Tab & Header UI**

**Why:** Macro UX first; buttons will call orchestrator.

**C++ Targets**

* SOptimizerPanel.h/.cpp
* Hook in MagicOptimizerModule.cpp (register tab + Tools menu)

**UI**

* Header: Deliverable (preset dropdown), Categories (Textures, Meshes, Materials, Runtime, World), Scope (Selection / Include / Exclude), Safety (Dry‑Run, Max Changes).
* Buttons: **Full Audit**, **Recommend**, **Apply**, **Verify**, **History…**

**Acceptance**

* Tab appears under Tools → MagicOptimizer; controls render; buttons are wired to no‑ops.

**Continue prompt**

Create SOptimizerPanel with a header row (preset dropdown, category checkboxes, scope, safety) and buttons (Full Audit, Recommend, Apply, Verify, History). Wire buttons to stub orchestrator calls.

**T1.3 Python bridge (C++ → Python)**

**Why:** One consistent callsite.

**C++ Targets**

* PythonBridge.h/.cpp

**Behavior**

* bool RunPython(const FString& Phase, const FString& Category, const FString& ConfigJsonPath, const FString& RunDir, FString& OutStdErr)
* Uses IPythonScriptPlugin to execute:
* import importlib, magic\_optimizer.entry as e; importlib.reload(e); e.run(r"<json>", phase="<phase>", category="<Category>", run\_dir=r"<dir>")

**Acceptance**

* Call to audit/all returns without error (Python stubs).

**Continue prompt**

Implement PythonBridge::RunPython using IPythonScriptPlugin. Build the command string to call magic\_optimizer.entry.run(json, phase, category, run\_dir) and propagate errors to logs.

**T1.4 Run orchestrator**

**Why:** Encapsulate the 4‑phase loop and history.

**C++ Targets**

* OptimizerRun.h/.cpp

**Behavior**

* Read UOptimizerSettings → compose FOptimizerConfig → SaveConfigJson.
* Execute audit for selected categories → recommend → (optional) apply → verify.
* Show modal summary: scanned/changed/errors per category; buttons to **Open Folder** or **Open CSV**.

**Acceptance**

* Clicking **Full Audit** creates <RunId> with audit CSVs (empty from stubs) + meta.json.

**Continue prompt**

Implement OptimizerRun::Execute(Mode) that writes config JSON, calls Python phases in order, and collects a JSON summary (counts + artifact paths) for a modal dialog.

**Milestone 2 — Textures (end‑to‑end)**

**T2.1 Python: textures/audit.py**

**Why:** Ground truth → CSV.

**Behavior**

* Scope: selection or include paths (fallback /Game).
* Fields (stable):  
  Path,Name,Class,LODGroup,Width,Height,NumMips,LODBias,CompressionSettings,SRGB,MipGenSettings,NeverStream,VirtualTextureStreaming,AddressX,AddressY,Filter,SourceFormat,ImportedWidth,ImportedHeight,HasAlpha,IsGreyscale,ReferencersTotal,ReferencersMaterials,ResourceSizeBytes

**Acceptance**

* Selecting ≥1 textures and clicking **Full Audit** writes textures\_audit.csv with ≥1 row.

**Continue prompt**

Implement textures/audit.py to enumerate Texture assets by selection or include paths and write textures\_audit.csv with the above columns. Guard referencer queries and missing fields.

**T2.2 Python: textures/recommend.py**

**Why:** Deterministic suggestions; reasons + flags.

**Behavior**

* Use **deliverable preset** caps & rules:
  + sRGB: true for color/emissive/UI; false for data maps.
  + Compression: Default/Normalmap/Masks/UI2D.
  + MipGen: NoMipmaps for UI/decals else FromTextureGroup.
  + Max size cap (Mobile\*, VR\*, PC\*).
  + Streaming: NeverStream for UI or ≤ threshold.
* Output textures\_recommendations.csv with:  
  Path,TypeGuess,Suggest\_SRGB,Suggest\_Compression,Suggest\_MipGen,Suggest\_MaxTextureSize,Suggest\_NeverStream,Suggest\_VirtualTextureStreaming,Suggest\_AddressX,Suggest\_AddressY,Suggest\_Filter,Change\_SRGB,Change\_Compression,Change\_MaxSize,Reason

**Acceptance**

* Running **Recommend** creates the CSV; Change\_\* reflects diffs vs audit.

**Continue prompt**

Implement textures/recommend.py that reads textures\_audit.csv, applies preset rules to propose new values, computes Change\_\* flags, and writes textures\_recommendations.csv.

**T2.3 Python: textures/apply.py**

**Why:** Touch only diffs; produce revert + apply CSV.

**Behavior**

* Load by **object path**; resolve enum names → UE enums.
* Apply only where different; save; collect diffs per asset.
* Append previous values to revert.json.
* Respect dry\_run, max\_changes, apply\_only, only\_selected.

**Output**

* textures\_apply.csv: Path,Applied\_Keys,Diff,Result,Error
* Update revert.json.

**Acceptance**

* Known wrong normal map becomes TC\_Normalmap & sRGB=false; apply CSV shows one row; revert exists.

**Continue prompt**

Implement textures/apply.py to read recommendations, resolve enums by name, set diffs, save assets, and write textures\_apply.csv. Support dry\_run, max\_changes, and collect a revert.json.

**T2.4 Python: textures/verify.py**

**Why:** Confidence; pass/fail + outliers.

**Behavior**

* Re‑audit; compare to recommendations; tolerate within caps.
* Output textures\_verify.csv; roll‑up verify\_report.json (pass/fail, counts).

**Acceptance**

* Verify shows **pass** for corrected assets; lists stragglers.

**Continue prompt**

Implement textures/verify.py to re‑audit, compare against recommended targets, write textures\_verify.csv, and return a summary JSON.

**T2.5 C++: Wire buttons → artifacts**

**Why:** UX loop complete.

**Behavior**

* Buttons call orchestrator with selected categories; open latest run folder; show counts with links to CSVs.

**Acceptance**

* You can **Audit → Recommend → Apply → Verify** textures in one click sequence; CSVs appear with correct headers.

**Continue prompt**

Connect the panel buttons to OptimizerRun::Execute with phase sequences. Add a modal summary with “Open Folder” and “Open CSV” actions for each artifact.

**Milestone 3 — Meshes (percent LODs)**

**T3.1 meshes/audit.py**

**CSV**  
Path,Name,LODGroup,Nanite,NumLODs,LOD0\_Tris,LOD1\_Tris,LOD2\_Tris,LOD3\_Tris

**T3.2 meshes/recommend.py**

* Name → category buckets; deliverable budgets for LOD1–3 triangles.
* Convert to **percent of LOD0**; clamp [0.0001, 1.0].

**T3.3 meshes/apply.py**

* Build StaticMeshReductionOptions with percents for LOD1–3; leave LOD0=1.0.
* Respect respect\_lod\_group; nanite\_default toggle is advisory at first.

**T3.4 meshes/verify.py**

* Re‑audit; ensure LOD1–3 triangles ≤ target.

**Acceptance (combined)**

* Known mesh reduces per preset; verify passes; no crash if Nanite meshes encountered.

**Continue prompt**

Implement meshes audit/recommend/apply/verify per UE 5.6 percent‑based reduction. Guard missing reduction interface and invalid LOD0.

**Milestone 4 — Materials (safe cleanups)**

**T4.1 materials/audit.py**

* Fields: Path,Name,InstructionCount,SamplerCount,UsesTranslucency,HasDeadStaticSwitches,MixedNormalSpaces,DuplicateSamplers

**T4.2 materials/recommend.py**

* Suggest StaticSwitch cleanup, sampler consolidation. Reasons only; default to **scan‑only**.

**T4.3 materials/apply.py (gated)**

* Only if apply\_only includes material\_cleanup; create diffs and save; slow path: preview first.

**T4.4 verify**

* Re‑audit counts down; record “changed” and “advised” separately.

**Milestone 5 — Runtime/World/Packaging (advice + toggles)**

* **runtime/** audit: scalability buckets, Lumen, VSM, AA, streaming pool/time slice, device profile. Recommendations per preset; applies through .ini and project settings where safe; otherwise **advice** rows only.
* **world/** audit: WP enabled, HLOD presence, Cull Distance Volumes; recommend HLOD generation flags (apply is deferred to Phase 2+).
* **packaging/** audit: pak compression, editor‑only excludes; recommendations per target.

**Acceptance**

* CSVs and verify JSONs exist; Apply touches only safe toggles (behind gates).

**Milestone 6 — History & Headless**

* History panel lists past runs; open artifacts.
* CLI/Commandlet or simple shell task: run audit/apply headless with config JSON.

**Milestone 7 — Polish**

* Per‑asset overrides import (CSV/JSON).
* Tooltips/hover docs; quick‑fix buckets.
* Revert last run (consume revert.json).

**Preset JSONs (starter templates)**

Create in Content/Python/magic\_optimizer/presets/:

* **Mobile\_AR.json** — cap\_color=1024, cap\_normal=512, cap\_mask=512; never\_stream\_small=256; lod\_target\_percent={1:0.33,2:0.18,3:0.10}; nanite=false; lumen=Off
* **PC\_High.json** — cap\_color=4096, normal=2048, masks=2048; TSR; Lumen=HWRT; VSM=On; streaming\_pool>=2048MB
* **Console.json** — balanced caps; Lumen=Software; dynamic res allowed.
* **VR.json** — aggressive caps; MSAA; no motion blur; bloom low; TAA off.
* **VirtualProduction.json** — high quality textures, Nanite on, Lumen HWRT, VSM; runtime tweaks are mostly advice.
* **Authoring\_Safe.json** — audit‑only defaults; zero apply except harmless texture flags.

**Cross‑cutting acceptance (global)**

* Every phase produces CSVs **with fixed headers** (even if 0 rows).
* meta.json stores CSV schema versions and preset label for diffability.
* Apply never writes without a matching recommendations CSV for the same run\_id.
* Verify always re‑audits and returns a **pass/fail** with outlier lists.

**Common Continue prompts (copy/paste)**

1. **Generate C++ settings + JSON I/O**

In Source/MagicOptimizer/, add OptimizerSettings.h/.cpp and OptimizerJson.h/.cpp. Implement UOptimizerSettings (EditorPerProjectUserSettings) and FOptimizerConfig with FJsonObjectConverter save/load to Saved/Optimizor/History/<RunId>/config.json. Include fields for preset, categories, scope, safety, gates, and per‑category configs.

1. **Create Python entry + dispatch**

In Content/Python/magic\_optimizer/entry.py, implement run(config\_path:str, phase:str, category:str, run\_dir:str) that loads JSON, ensures run\_dir, and dispatches to category modules’ audit/recommend/apply/verify. Return a dict with counts and artifact paths.

1. **Textures audit (full columns)**

Implement textures/audit.py that scans selection or include paths and writes textures\_audit.csv with the full schema (Path, Name, Class, LODGroup, Width, Height, NumMips, LODBias, CompressionSettings, SRGB, MipGenSettings, NeverStream, VirtualTextureStreaming, AddressX, AddressY, Filter, SourceFormat, ImportedWidth, ImportedHeight, HasAlpha, IsGreyscale, ReferencersTotal, ReferencersMaterials, ResourceSizeBytes).

1. **Textures recommend/apply/verify**

Implement textures/recommend.py, textures/apply.py, and textures/verify.py using preset rules and enum‑name resolution. Write textures\_recommendations.csv, textures\_apply.csv, and textures\_verify.csv respectively; support dry\_run, max\_changes, apply\_only.

1. **Meshes percent LODs**

Implement meshes/audit.py (LOD counts and tris), meshes/recommend.py (convert target tris to percent of LOD0; clamp), meshes/apply.py (SetLODs with percents), and meshes/verify.py.

**Notes / Risks to watch**

* **UE 5.6 percent‑only mesh reduction** (no max\_triangles)—always convert from triangle budgets.
* **Object path vs name**—store and resolve **by object path** in CSVs.
* **Source control**—if enabled, guard with auto‑checkout toggle; still save gracefully when off.
* **Editor state**—close editors before writes; re‑load assets post‑save if Details panel should reflect changes.

You can now paste tasks into Continue’s **Plan** and run them sequentially. If you want, I can generate **starter code files** for T1.1–T1.4 and T2.1 immediately so you can compile and run the first **Full Audit**.