**In one paragraph (what this is)**

AstroLotto is a Streamlit app that loads lottery history from CSVs, builds features, trains tabular ML models, and suggests number picks for several games. It decorates the UI with “Cosmic Conditions” and local weather, keeps all data files in a Data/ folder (with automatic backups), and can train on CPU by default or use GPU/Apple MPS if available. You launch it with a one-file script; it sets paths, creates the venv, installs deps, and runs the app.

**Folder layout (current)**

css

CopyEdit

AstroLotto\_v10.6/

Program/ ← all Python code (app + utilities + models folder)

Data/ ← all CSVs (historical draws, predictions, cached files)

backups/ ← automatic CSV backups

Extras/ ← venv, requirements.txt, .streamlit, bsp, prefs (not in zip due to size)

RUN\_AstroLotto.bat ← Windows launcher

RUN\_AstroLotto.command ← macOS launcher (if you added it)

**How it starts (launch → app)**

1. **Launcher**
   * Windows: RUN\_AstroLotto.bat
   * macOS: RUN\_AstroLotto.command  
     What it does: checks Python 3.11, creates Extras/.venv311 if needed, installs Extras/requirements.txt, sets env vars so the app can find Program/, Data/, Extras/, then runs streamlit run Program/app\_main.py.
2. **sitecustomize.py boots** (auto-imported)  
   Program/sitecustomize.py runs very early and wires up the environment:
   * Sets thread caps so the box doesn’t melt.
   * **Installs CSV routing** (see below).
   * Patches **weather** and **astrology** helpers to be tolerant and always return a summary string.
   * Adds a **backfill shim** so the app can call run\_backfill\_for\_csv(...) regardless of how historical\_backfill was authored.
   * Patches Powerball/Mega modules to:
     + Render the **Cosmic + Weather** panel automatically.
     + Use a **robust prediction normalizer** (fixes the “list → int” type error).
3. **Streamlit app**  
   Program/app\_main.py builds the sidebar/pages, loads settings, and renders each game tab (Powerball, Mega Millions, Colorado Lotto+, Cash 5, Pick 3, Lucky for Life, etc.). Each tab:
   * Loads historical draws from the corresponding CSV in Data/.
   * Calls feature builders and the predictor.
   * Shows **Cosmic Conditions** and **Weather** for that tab.
   * Outputs suggested picks and can write new CSVs (predictions, caches).

**The moving parts (what calls what)**

bash

CopyEdit

Launcher (.bat/.command)

└─ streamlit run Program/app\_main.py

├─ utilities/io\_routing.install() (sitecustomize)

├─ utilities/astrology + weather (patched by sitecustomize)

├─ utilities/ui\_cosmic.render\_cosmic\_weather\_panel()

├─ historical\_backfill.\* (called via backfill shim)

├─ utilities/features.build\_features()

├─ autogluon.tabular.TabularPredictor (training/inference)

└─ CSV I/O → Data/ + Data/backups/ (routed)

**Program core**

* **app\_main.py**  
  Entry point that wires Streamlit navigation and pulls in each **game page** (\*\_predictor.py or pages package).
* **Game predictors** (e.g., powerball\_predictor.py, mega\_millions\_predictor.py, etc.)
  + Load CSVs from Data/….
  + Build a training set using utilities/features.py (build\_features, plus any helper constants like GAME\_MAP).
  + Train or load an **AutoGluon Tabular** model and generate candidate picks.
  + Normalize output to { white: [..], special: n }.
  + Render picks, **Cosmic Conditions**, **Weather**, and status text.
* **utilities/**
  + **io\_routing.py**: intercepts open() and directory creation so:
    - All **writes** of .csv go to **Data/**.
    - All **backups** go to **Data/backups/**.
    - **Reads** fall back to Data/ if code asks elsewhere.
    - Also redirects os.makedirs/Path.mkdir to stop stray folders under root/Program.
  + **astrology.py**: moon position, phases, alignment score, and a helper to **derive the user’s zodiac sign** from birthday. sitecustomize makes sure missing bits don’t crash.
  + **weather.py**: fetches basic weather for a ZIP. sitecustomize shims extra args and guarantees a summary field.
  + **ui\_cosmic.py**: draws the combined **Cosmic + Weather** panel in the UI.
  + **features.py**: turns raw draws into model-ready features. Typical stuff: lag/recency, hot/cold counts, day-of-week/month, maybe rolling windows; some versions include cosmic or calendar features.
  + **backfill\_runner\_helper.py**: tiny glue; sitecustomize provides a fallback if your historical module uses different function names.
  + **historical\_backfill.py**: game-specific scraping/fill logic used to complete or refresh historical CSVs.
* **models/**  
  Folder where AutoGluon saves trained predictors per game. The sitecustomize patch auto-appends a timestamp if the path exists, avoiding overwrite warnings.

**Data files (the CSVs)**

* **Historical**: Data/cached\_powerball\_data.csv, cached\_megamillions\_data.csv, etc.  
  One row per draw. The predictor expects a consistent date column (standardized to draw\_date) and game columns (numbers + special/bonus).
* **Predictions**: Data/\*\_predictions.csv  
  Where suggested picks get written, with timestamps and any metadata you chose to log.
* **Backups**: Data/backups/\*  
  Every save creates a dated backup before overwrite. The router ensures **no** backups/ appears in the root.

**Backfill flow**  
User hits “refresh history” or a backfill button → tab calls backfill\_runner\_helper.run\_backfill\_for\_csv(path, game=...) → the shim detects and calls a function inside historical\_backfill.py that actually fills the file. It never throws if the function signature differs; it tries variants and reports what worked.

**The model (how the AI part works)**

* **AutoGluon Tabular** trains an ensemble stack of tabular learners. Out of the box you’ll see **LightGBM**, **XGBoost**, **CatBoost** do the heavy lifting.
* If **PyTorch + fastai** are installed (and on Mac, MPS is enabled), neural nets join the party. If **Transformers** are present, AutoMM can activate for richer modeling. If **Ray** is installed, folds/HPO parallelize.
* We added two quality-of-life fixes in sitecustomize:
  1. **Sanitize dtypes**: pandas nullable ints and list-like objects are made numeric / string-safe before training to silence the Int64/Numpy warnings and avoid quirky crashes.
  2. **Unique model path**: if Program/models/<name> already exists, a timestamp is added to avoid the “may overwrite” warning.
* Output from different models can be messy (lists, strings, nested arrays). The **robust normalizer** standardizes it to:

css

CopyEdit

[{"white": [w1, w2, w3, w4, w5], "special": s}]

with correct ranges per game (PB 1–69 + PB 1–26; Mega 1–70 + Mega 1–25).

**Feature logic** (high level)

* From historical draws, the features builder typically computes:
  + **Recency/frequency**: how often each number or combination shows up in rolling windows.
  + **Temporal**: day-of-week, month, gaps since last seen, streaks.
  + **Game-specific** constraints and derived stats.
  + Optional **cosmic/calendar** inputs depending on availability.  
    Exact columns vary by game, but the goal is to give the tree models useful signals without breaking lottery rules.

**Cosmic Conditions & Weather (UI logic)**

* Each tab calls **ui\_cosmic.render\_cosmic\_weather\_panel()** which shows:
  + A **summary** of planetary alignment / moon position from astrology.py.
  + A **weather** summary by ZIP.
* sitecustomize ensures both functions are tolerant:
  + Weather accepts extra args and returns a human-readable summary.
  + Astrology guarantees get\_user\_sign\_from\_birthday exists; moon/align calls return safe summaries even if an underlying call fails.

**CSV routing & backups (why nothing leaks into root/Program)**

* Any code that writes a .csv—even if it tries Program/…, csv/…, or plain predictions.csv—gets intercepted and **rewritten to Data/…**.
* Any directory creation that looks like backups/, csv/, datasets/, or Program/… for CSVs is **redirected under Data/**.
* Reads that point at the old spots **fall back to Data/** so nothing breaks.  
  Result: clean root, clean Program, everything lives in **Data/** with automatic **Data/backups/**.

**Performance and hardware**

* Defaults to CPU. Thread caps are set to 4 in env to keep your machine responsive.
* **Windows GPU**: run your Enable\_GPU\_Extras\_v2.bat to install Torch, fastai, Transformers, Ray into Extras/.venv311.
* **macOS MPS**: run Enable\_MPS\_Extras.command. Torch’s MPS backend accelerates NN models on Apple Silicon.

**Typical end-to-end flow**

1. Launch. App creates venv if needed, installs deps, starts Streamlit.
2. Pick a game tab.
3. If history is missing or stale, hit **Refresh/Backfill** → file updates in Data/… and Data/backups/….
4. App builds features → trains or loads a model → generates candidate picks.
5. Picks are normalized and displayed. Predictions can be saved to CSV.
6. **Cosmic + Weather** panel renders for context.
7. Repeat for other games.

**Common gotchas we already guarded against**

* **Wrong CSV locations / root backups** → io\_routing v3.3 fixes future writes and folder creation.
* **Import shape drift** (get\_user\_sign\_from\_birthday, build\_features) → shims and soft checks added.
* **PB/MM prediction type errors** → robust normalizer.
* **Double browser windows** or bad flags → fixed launchers.
* **AutoGluon warnings** (Int64 dtype, overwrite paths, missing torch/fastai/ray) → dtype sanitizer, unique paths, conditional model exclusion until the libs are installed.

**How to extend safely**

* **Add a new game**: make a new \*\_predictor.py, add a CSV name in Data/, reuse features.build\_features, and register the page. The router will handle files.
* **Change features**: edit utilities/features.py only; predictors shouldn’t need changes.
* **Add data sources**: enhance historical\_backfill.py; the helper shim will call into it without changing tabs.