BLM

1. **Load configuration and input paths (main block)**
    - Read `config.yaml` to get BLM input GDB path, layer name, reference GDB path, start/end years, and output GDB/layer name.
    - Call `enrich_BLM(...)` with these parameters.
2. **Load BLM treatments into a GeoDataFrame**
    - Try to read a shapefile directly with `gpd.read_file(blm_gdb_path)`.
    - If successful, rename short field names (`SYS_TRTMNT`, `TRTMNT_TYP`, etc.) to the standard BLM field names.
    - If that fails, read from a file geodatabase layer using `OpenFileGDB` and SQL.
    - Log the load time.
3. **Validate schema and reproject**
    - Check that all required `BLM_COLUMNS` exist using `verify_gdf_columns`.
    - Reproject geometries to EPSG 3310.
    - Log/show columns for inspection.
4. **Standardize spatial extent and geometry**
    - **Clip** all BLM features to California using `clip_to_california(...)`.
    - **Repair** invalid geometries with `repair_geometries(...)`.
5. **Add Task Force common columns**
    - Use `add_common_columns(...)` to attach the standard Task Force schema fields.
    - Log/show the standardized columns.

6. **Populate core project and agency fields**
    - Set IDs and admin metadata:
        - `PROJECTID_USER = UNIQUE_ID`
        - `AGENCY = "DOI"`
        - `ORG_ADMIN_p/t/a = "BLM"`
        - `ADMINISTERING_ORG = "BLM"`
        - `PROJECT_NAME = TRTMNT_NM`
        - `IMPLEMENTING_ORG = "BLM"`
        - `ACTIVITY_NAME = TRTMNT_NM`
        - `PRIMARY_FUNDING_SOURCE = "FEDERAL"`
        - `PRIMARY_FUNDING_ORG = "NPS"`
        - `BVT_USERD = "NO"`
7. **Parse start and end dates into activity dates**
    - Define `safe_date_convert(...)` to handle multiple string formats and fix years starting with `"02"` in `TRTMNT_START_DT` and `TRTMNT_END_DT`.
    - Create `ACTIVITY_START` and `ACTIVITY_END` by applying this converter.
8. **Derive activity status and treated acres**
    - Set `ACTIVITY_STATUS = "COMPLETE"` for all records.
    - Compute `ACTIVITY_QUANTITY` in acres using helper `ifelse(BLM_ACRES, GIS_ACRES)`:
        - Use `GIS_ACRES` if `BLM_ACRES` is `0` or `NaN`, otherwise use `BLM_ACRES`.

- Cast `ACTIVITY_QUANTITY` to `float` and set `ACTIVITY_UOM = "AC"`.

9. **Fill organization and source fields, compute year**
    - Set:
        - `ADMIN_ORG_NAME = "BLM"`
        - `IMPLEM_ORG_NAME = "BLM"`
        - `PRIMARY_FUND_SRC_NAME = "FEDERAL"`
        - `PRIMARY_FUND_ORG_NAME = "BLM"`
        - `Source = "BLM"`
    - Derive `Year` from `ACTIVITY_END.year` (or `None` if no end date).
10. **Apply three-pass activity Crosswalk logic**
    - **Crosswalk rule 1 (type + subtype → standard category):**
        - Map combinations of `TRTMNT_TYPE_CD` and `TRTMNT_SUBTYPE` to values like `PRESCRB_HERBIVORY`, `NOT_DEFINED`, `PEST_CNTRL`, `BROADCAST_BURN`, `THIN_MECH`, `HABITAT_REVEG`, otherwise keep existing `Crosswalk`.
    - **Crosswalk rule 2 (treatment name keywords):**
        - Based on `TRTMNT_NM` and type/subtype, set `Crosswalk` to `PILE_BURN`, `ROAD_CLEAR`, `CHIPPING`, `THIN_MAN`, or `MASTICATION` when those keywords appear.
    - **Crosswalk rule 3 (comments keywords):**
        - Based on `TRTMNT_COMMENTS` and type/subtype, refine `Crosswalk` to `PILE_BURN`, `BROADCAST_BURN`, `THIN_MAN`, `CHIPPING`, `LOP_AND_SCAT`, `MASTICATION`, `MOWING`, `BIOMASS_REMOVAL`, or `PILING`.

11. **Filter treatments by analysis years**
    - Keep only rows where `Year` is between `start_year` and `end_year` (inclusive) in `selected_gdf`.
12. **Align with Task Force template and set geometry type**
    - Create an empty GeoDataFrame `new_blm` using `get_wfr_tf_template(...)` columns and EPSG:3310.
    - Append `selected_gdf` into this template.
    - Set `TRMT_GEOM = "POLYGON"` for all records.
13. **Drop non-standard fields and enrich with reference layers**
    - Use `keep_fields(...)` to remove any columns not in the Task Force standard.
    - Log/show cleaned columns.
    - Call `enrich_polygons(...)` to add vegetation, ownership, county, WUI, Task Force Region, and year attributes from the reference geodatabase, resulting in `enriched_blm`.
14. **Construct user-facing treatment IDs**
    - Build `TRMTID_USER` as a string composed of:
        - `PROJECTID_USER` first 7 chars
        - `COUNTY` first 3 chars
        - `PRIMARY_OWNERSHIP_GROUP` first 4 chars
        - `IN_WUI` first 3 chars
        - `PRIMARY_OBJECTIVE` first 8 chars
15. **Assign domains and apply MAS cutoff rule**

- Run `assign_domains(...)` to attach coded value domains.
- For rows with `ACTIVITY_END >= '2024-10-01'`, set `COUNTS_TO_MAS = "NO"`.

16. **Save enriched BLM treatments**
    - Write the final `enriched_blm` GeoDataFrame to the output geodatabase and layer using `save_gdf_to_gdb(...)`.
    - In `__main__`, log memory usage at the end of the run.

CNRA

1. **Load and validate CNRA input layers**
    - Read CNRA polygon, line, point, project, and activity layers from the input geodatabase.
    - Verify each layer has the expected columns.
    - Reproject polygons/lines/points/projects to EPSG:3310 and uppercase their column names; uppercase activity table columns.
2. **Prepare a unified CNRA activity table** (`prepare_activity_table`)
    - Derive a single `TREATMENTID_` for each activity from `TREATMENTID_LN`, `TREATMENTID_PT`, or `TREATMENTID_POLY`.
    - Drop activities with missing `TREATMENTID_`.
    - Convert `ACTIVITY_START` and `ACTIVITY_END` to dates (removing milliseconds).
    - Create an empty standardized activity schema and append only columns that exist in both the schema and the original activity table.

3. **Prepare CNRA project table** (`prepare_project_table`)
    - For projects, change `AGENCY` from `"CALFIRE"` to `"CNRA"` where needed.
    - Set `PROJECTID_USER = GlobalID + "-CNRA"` for non-null `GlobalID`.
    - Drop duplicate projects based on `PROJECTID_USER`.
4. **Join features with activities (per geometry type)** (`enrich_CNRA_features`)
    - For polygons/lines/points, drop features with null `GlobalID`.
    - Merge features with the prepared activity table using `GlobalID` (features) = `TREATMENTID_` (activities).
    - Set `TRMTID_USER = GlobalID + "-CNRA"` and drop duplicate merged rows.
5. **Join with project information**
    - Merge the feature+activity table with the project table using `PROJECTID` (features/activities) = `GlobalID` (projects).
    - Set `PROJECTID_USER = PROJECTID + "-CNRA"` where `PROJECTID` is not null.
6. **Create standardized feature GeoDataFrame**
    - Build an empty standardized feature class from the Task Force template (`get_wfr_tf_template`).
    - Select the intersection of columns between the CNRA flat table and the template plus geometry.
    - Construct `standardized_features` GeoDataFrame with the template's CRS.
7. **Standardize and enrich core CNRA attributes** (`enrich_standardized_features`)
    - Set `Crosswalk = ACTIVITY_DESCRIPTION`.
    - Set `Source = "CNRA"`.

- Fill `ORG_ADMIN_a`, `ORG_ADMIN_p`, and `ADMINISTERING_ORG` from `ORG_ADMIN_t` where they are missing.
- Default missing `AGENCY` to `"CNRA"`.
- Default missing `ACTIVITY_STATUS` to `"COMPLETE"`.
- If `ACTIVITY_END` is missing:
  - For `ACTIVE/Active/COMPLETE/Complete`, use the current datetime.
  - For `PLANNED/Planned`, use `ACTIVITY_START` if available, otherwise current datetime.
- Call `add_common_columns` to attach standard Task Force fields.

8. **Geometry-specific handling and admin clean-up**
   - For `feature_type == "point"`, ensure geometries are points by taking centroids.
   - Recode `ADMINISTERING_ORG` from `"MRCA"` to `"SMMC"` where applicable.
9. **Enrich with vegetation, ownership, county, WUI, region, etc.**
   - Trim to standard fields via `keep_fields`.
   - Call the appropriate enrichment:
     - `enrich_polygons` for polygon features.
     - `enrich_lines` for line features.
     - `enrich_points` for point features.
   - Trim again with `keep_fields`, then apply `assign_domains` to map values to domain-coded fields.
10. **Save enriched outputs**
    - Write enriched polygon, line, and point feature classes to the output geodatabase using `save_gdf_to_gdb`, naming each layer with the base

`output_layer_name` plus `_polygon`, `_line`, or `_point`.

11. **Main configuration-driven execution**
    - Read paths, layer names, and `start_year`/`end_year` from `config.yaml`.
    - Construct formatted output geodatabase path and layer name.
    - Call `enrich_CNRA` once to process polygons, lines, and points.
    - Log final process memory usage in MB.

CALTRANS

1. **Initialize logging and suppress warnings**
    - Configure a named logger (`enrich.enrich_CalTrans`) and suppress verbose `pyogrio`, `RuntimeWarning`, `FutureWarning`, and measured-geometry warnings.
2. **Define expected Caltrans schemas**
    - Declare required column lists for **road/tree activity** and **road/tree treatment** tables to validate inputs later.
3. **Helper functions for area and units**
    - `calc_treatment_area(uom, quantity)`: if units are `"ACRE"` or `"AC"`, return the quantity as treatment area; otherwise return `None`.
    - `convert_uom(uom)`: convert `"ACRE"` to `"AC"`, otherwise leave the unit unchanged.
4. **Read and normalize Caltrans road activity data** (`enrich_Caltrans`)
    - Load the **road activity** layer from the input geodatabase using `gpd.read_file`.
    - Rename activity fields to the ITS/Task Force naming using `activity_dict`.

- If the `Route` field is missing, create it from `Route_Num`, left-padding with zeros to a consistent width.
- Validate that the activity table contains all `CALTRANS_ROAD_ACTIVITY_COLUMNS` and log its columns.

5. **Read and normalize Caltrans road treatment data** (`enrich_Caltrans`)
    - Load the **road treatment** layer from the input geodatabase.
    - Rename treatment fields using `treatment_dict` to match the expected schema.
    - Validate against `CALTRANS_ROAD_TREATMENT_COLUMNS`.
    - Reproject treatment geometries to EPSG:3310 and log the columns.
6. **Merge activity and treatment data** (`caltrans`)
    - Log that standardization is starting.
    - **Step 1/10**: Inner-join `caltrans_lines` (treatments) and `caltrans_table` (activities) on `['Highway_ID', 'Calendar_Year']`.
    - Log the number of merged records.
7. **Validate geometry**
    - **Step 2/10**: Count invalid geometries in `merged_data.geometry`.
    - If any invalid geometries are found, log an error and terminate (`exit()`).
8. **Add Task Force common columns**
    - **Step 3/10**: Call `add_common_columns(merged_data)` to attach the standard Task Force fields.
9. **Populate project, treatment, and activity attributes**
    - **Step 4/10**: Set core fields, including:

- `PROJECTID_USER = Highway_ID`
- `AGENCY = 'CALSTA'`
- All admin org fields (`ORG_ADMIN_p/t/a`, `ADMINISTERING_ORG`, `ADMIN_ORG_NAME`) to `'CALTRANS'`
- Project contact and email (`'Division of Maintenance'`, `'andrew.lozano@dot.ca.gov'`)
- Funding source/org fields (`GENERAL_FUND`, `CALTRANS`)
- `TRMTID_USER` as a concatenation of `Highway_ID`, `From_PM_C`, and `To_PM_C`.
- `TREATMENT_AREA` using `calc_treatment_area(UOM, Production_Quantity)`.
- `ACTIVID_USER` as `"CALTRANS-{Work_Order_Number}-{row_index}"`.
- `IMPLEMENTING_ORG` and `IMPLEM_ORG_NAME` from `District`.
- `ACTIVITY_UOM` as standardized units via `convert_uom(UOM)`.
- `ACTIVITY_QUANTITY = Production_Quantity`.
- `ACTIVITY_STATUS = 'COMPLETE'`.
- `ACTIVITY_START = WorkBeginDate`, `ACTIVITY_END = WorkEndDate`.
- `Source = 'CALTRANS'`.
- `Crosswalk = Activity_Description`.
- `TRMT_GEOM = 'LINE'`.

10. **Trim to standard schema and enrich lines**

- **Step 5/10**: Use `keep_fields` to reduce `merged_data` to the Task Force standard columns.

- **Step 6/10**: Call `enrich_lines(merged_data, a_reference_gdb_path, start_year, end_year)` to derive vegetation, region, WUI, ownership, etc., producing `enriched_data`.

11. **Set ownership and recompute treatment ID**

    - **Step 7/10**: Set `PRIMARY_OWNERSHIP_GROUP = 'STATE'`.

    - Rebuild `TRMTID_USER` as:
      `PROJECTID_USER` + COUNTY (first 8 chars) + REGION (first 3 chars) + IN_WUI (first 3 chars), joined with hyphens.

12. **Final schema cleanup and domain assignment**

    - **Step 8/10**: Call `keep_fields(enriched_data)` again to remove unnecessary columns.

    - **Step 9/10**: Apply `assign_domains(enriched_data)` to map values to domain-coded fields.

13. **Save enriched Caltrans line treatments**

    - **Step 10/10**: Save the final `enriched_data` to the output file geodatabase and layer via `save_gdf_to_gdb(..., group_name="c_Enriched")`.

14. **Config-driven main execution and logging**

    - In `__main__`, load paths, layer names, and year range from `config.yaml`.

    - Build formatted output geodatabase path and layer name using `start_year`, `end_year`, and today's date.

    - Call `enrich_Caltrans(...)` once to process the road activity/treatment inputs.

    - After processing, compute and log the process memory usage in MB.

IFPRS

1. **Load IFPRS treatments**
    - Read the IFPRS feature class from the input geodatabase into a GeoDataFrame and verify it contains all required `IFPRS_COLUMNS`.
2. **Filter to relevant IFPRS records**
    - Keep only records where `State == 'California'`.
    - Convert `Completion` to UTC datetimes and keep only records with `Completion >= 2023-10-01`.
    - Reproject geometries to EPSG:3310.
3. **Standardize spatial extent and geometry**
    - **Step 1/15**: Clip features to California using `clip_to_california`.
    - **Step 2/15**: Repair invalid geometries using `repair_geometries`.
4. **Add Task Force common schema**
    - **Step 3/15**: Call `add_common_columns` to attach the standard Task Force fields.
    - Log the resulting columns and shape.
5. **Populate project and agency attributes**
    - **Step 4/15**: Set:
        - `PROJECTID_USER`, `PROJECT_NAME`, `ACTIVITY_NAME` from `Name`.
        - `AGENCY = "DOI"`.
        - `ORG_ADMIN_p/t/a`, `ADMINISTERING_ORG`, `ADMIN_ORG_NAME`, `IMPLEM_ORG_NAME` from `Agency`.

        - `PRIMARY_FUNDING_SOURCE = "FEDERAL"` and `PRIMARY_FUNDING_ORG`, `PRIMARY_FUND_ORG_NAME`, `IMPLEMENTING_ORG` from `Agency`.

        - `BVT_USERD = "NO"`.

6. **Convert initiation and completion dates**

    - **Step 5/15**: Use `safe_date_convert` to create `ACTIVITY_START` from `Initiation` and `ACTIVITY_END` from `Completion`, with robust error handling.

7. **Map IFPRS status to Task Force activity status**

    - **Step 6/15**: Map `Status` values (`started`, `not started`, `completed`, `cancelled`) to `ACTIVITY_STATUS` (`Active`, `Planned`, `Complete`, `Cancelled`), defaulting to `"Unknown"` for anything else.

8. **Compute activity quantity and units**

    - **Step 7/15**:

        - Copy `Calculated` into `TotalAcres`.

        - For each record, set `ACTIVITY_QUANTITY` to `TotalAcres` if non-zero; otherwise use the polygon area (`Shape_Area / 4046.86` acres).

        - Cast `ACTIVITY_QUANTITY` to float and set `ACTIVITY_UOM = "AC"`.

9. **Set funding/source metadata and derive year**

    - **Step 8/15**:

        - Set `ADMIN_ORG_NAME`, `IMPLEM_ORG_NAME`, `PRIMARY_FUND_SRC_NAME = "FEDERAL"`, `PRIMARY_FUND_ORG_NAME`, and `Source = "IFPRS"`.

        - Derive `Year` from `ACTIVITY_END.year`.

10. **Assign activity crosswalk codes**

- **Step 9/15**: Compute `Crosswalk` by applying `crosswalk_activity(Type, SubType, Name, Notes)` which:
    - Maps specific `Type` values (e.g., `broadcast`, `pile burn`, `thinning`, `herbicide`, `planting`) to Task Force activity codes (e.g., `BROADCAST_BURN`, `PILE_BURN`, `THIN_MECH`, `HERBICIDE_APP`).
    - Uses `Name` and `Notes` text patterns (e.g., `"pile"`, `"broadcast"`, `"lop"`, `"masticat"`) to further classify treatments.
    - Defaults to `"NOT_DEFINED"` when no rule matches.

11. **Filter by analysis years and conform to template**
    - **Step 10/15**: Keep only records where `Year` is between `start_year` and `end_year`.
    - **Step 11/15**: Create an empty GeoDataFrame using `get_wfr_tf_template`.
    - **Step 12/15**: Append the selected IFPRS records to this template.
    - **Step 13/15**: Set `TRMT_GEOM = "POLYGON"` for all records.
12. **Trim to standard fields and enrich polygons**
    - **Step 14/15**: Use `keep_fields` to drop non-standard fields and log the resulting schema.
    - **Step 15/15**: Call `enrich_polygons` to add vegetation, ownership, county, WUI, Task Force Region, and related attributes.
13. **Create user treatment IDs and assign domains**
    - **Step 16/15**: Build `TRMTID_USER` as: `PROJECTID_USER + "-" + COUNTY[:3] + "-" + PRIMARY_OWNERSHIP_GROUP[:4] + "-" + IN_WUI[:3] + "-" + PRIMARY_OBJECTIVE[:8]`.
    - **Step 17/15**: Run `assign_domains` to apply domain mappings.

- Set `AGENCY = "DOI"` again as a temporary fix.

14. **Save enriched IFPRS output and log memory usage**

    - **Step 18/15**: Save the enriched IFPRS layer to the output geodatabase and layer name using `save_gdf_to_gdb(..., group_name="c_Enriched")`.

    - In `__main__`, load paths and parameters from `config.yaml`, call `enrich_IFPRS(...)`, and log the Python process memory usage in MB.

NFPORS

1. **Load NFPORS inputs from config and file**

    - Read the polygon, BIA point, and (optionally) FWS point layers from the NFPORS geodatabase.

    - Normalize column names (lowercase + remap some abbreviated field names) and verify each layer has the expected schema (`NFPORS_POLYGON_COLUMNS`, `NFPORS_BIA_COLUMNS`, `NFPORS_FWS_COLUMNS`).

    - Reproject all GeoDataFrames to EPSG:3310 and log their columns.

2. **Filter and standardize NFPORS polygon features**

    - Keep only polygon records where `agency` is `BIA` or `FWS`.

    - Filter to records with `act_comp_dt >= 1995-01-01` or `act_comp_dt` is null.

    - Keep only polygons where `st_abbr == 'CA'`.

    - Repair invalid polygon geometries.

3. **Map polygon attributes into the Task Force schema**

    - Rename `agency` → `agency_` and call `add_common_columns` to attach the standard Task Force fields.

    - Populate core fields:

        - `PROJECTID_USER = 'NFPORS' + trt_id`
        - `AGENCY = 'DOI'`
        - `ADMINISTERING_ORG` and `IMPLEMENTING_ORG` from `agency_`
        - `ACTIVITY_NAME` from `trt_nm`
        - `ACTIVITY_UOM = 'AC'`
        - `ACTIVITY_QUANTITY` from `gis_acres`
        - `ACTIVITY_START` from `modifiedon`
        - `ACTIVITY_END` as `act_comp_dt` if present, otherwise `modifiedon`
    - Derive `ACTIVITY_STATUS` from `trt_statnm` (`Accomplished → COMPLETE`, `Initiated → ACTIVE`, otherwise keep original).
    - Set `Source = 'nfpors_haz_fuels_treatments_reduction'` and `Crosswalk = type_name`.
4. **Standardize polygon fields and enrich**
    - Use `keep_fields` to restrict to the Task Force-standard output columns.
    - Call `enrich_polygons(...)` to attach reference-based attributes.
    - Build `TRMTID_USER` as a concatenation of:
        - last 7 chars of `PROJECTID_USER`, `IN_WUI` (first 3 chars), `PRIMARY_OWNERSHIP_GROUP` (first char), `COUNTY` (first 8 chars), and `PRIMARY_OBJECTIVE` (first 12 chars).
    - Run `assign_domains(...)` to apply coded-value/domain mappings.
5. **Correct polygon activity quantity and MAS flag**

- Overwrite `ACTIVITY_QUANTITY` with `TREATMENT_AREA` to correct multipart FWS polygon over-counting.
- Set `AGENCY = 'DOI'` (explicitly).
- For polygons with `ACTIVITY_END >= '2023-10-01'`, set `COUNTS_TO_MAS = 'NO'`.
- Save the enriched polygon layer as `<output_layer_name>_polygon` using `save_gdf_to_gdb`.

6. **Filter and combine NFPORS point features (BIA/FWS)**
   - For BIA points, normalize and remap abbreviated column names to the expected schema.
   - If a separate FWS layer is not provided, create an empty FWS DataFrame with the same columns as BIA.
   - Filter BIA and FWS to `statename == 'California'` and concatenate them into `combined_pts`.
7. **Map point attributes into the Task Force schema**
   - Rename selected fields (`source → source_`, `projectid → project_id`, `latitude → latitude_`, `longitude → longitude_`) and call `add_common_columns`.
   - Populate core fields:
     - `PROJECTID_USER = 'NFPORS' + project_id`
     - `AGENCY = 'DOI'`
     - `ADMINISTERING_ORG` and `IMPLEMENTING_ORG` from `agencyname`
   - Convert WUI flags: `iswui == 'Y' → 'WUI_USER_DEFINED'`, `iswui == 'N' → 'NON-WUI_USER_DEFINED'`, stored in `IN_WUI`.
   - Set:

        - `ACTIVITY_NAME` from `treatmentname`
        - `ACTIVITY_UOM` from `unitofmeas`
        - `ACTIVITY_QUANTITY` = `totalaccomplishment` if nonzero, otherwise `plannedaccomplishment`
        - `TREATMENT_AREA = ACTIVITY_QUANTITY` when `ACTIVITY_UOM == 'AC'`, else `None`.

8. **Derive point dates and status**
    - Compute `ACTIVITY_START` using `handle_date(actualinitiationdate, plannedinitiationdate)`, treating dates before 1901-01-01 as invalid and falling back to planned dates.
    - Set `ACTIVITY_END = actualcompletiondate`.
    - Initialize `ACTIVITY_STATUS = 'PLANNED'`, then:
        - Set to `'ACTIVE'` if `actualinitiationdate` is a valid date (>= 1901-01-01).
        - Set to `'COMPLETE'` where `iscompleted == '1'`.
    - Set `Source = 'nfpors_current_fy_treatments'`, `Crosswalk = typename`, and `TRMT_GEOM = 'POINT'`.

9. **Standardize point fields and enrich**
    - Use `keep_fields` to keep only standard Task Force columns.
    - Call `enrich_points(...)` to attach reference-based attributes.
    - Build `TRMTID_USER` as:
        - last 7 chars of `PROJECTID_USER`, `IN_WUI` (first 3 chars), `PRIMARY_OWNERSHIP_GROUP` (first char), `COUNTY` (first 8 chars), and `PRIMARY_OBJECTIVE` (first 12 chars).

- Run `assign_domains(...)` to apply domain mappings.

10. **Apply MAS cutoff and save points**
    - For points with `ACTIVITY_END >= '2023-10-01'`, set `COUNTS_TO_MAS = 'NO'`.
    - Save the enriched point layer as `<output_layer_name>_point` using `save_gdf_to_gdb`.
11. **Driver and logging**
    - In `__main__`, read all paths, layer names, years, and output formats from `config.yaml`, call `enrich_NFPORS(...)`, and log the process's memory usage in MB.

NPS

1. **Load configuration and inputs**
    - Read paths, layer names, start/end years, and output formats from `config.yaml`.
    - Decide input source:
        - If an ArcGIS feature URL is provided, download NPS treatments via `fetch_arcgis_feature_service`.
        - Otherwise, read from the local file (`shp` or `OpenFileGDB`), remapping abbreviated column names when needed.
2. **Validate and prepare raw NPS data**
    - Verify that the input GeoDataFrame contains all required `NPS_COLUMNS`.
    - Reproject geometries to `EPSG:3310`.
    - Log and inspect the columns.
3. **Filter by completion date**

- Convert `ActualCompletionDate` to `datetime`.
- Keep only records with `ActualCompletionDate > '1995-01-01'` or `ActualCompletionDate` missing.

4. **Repair geometries and clip to California**
   - Repair invalid geometries using `repair_geometries`.
   - Clip the treatments to California using `clip_to_california` with the reference geodatabase.
5. **Dissolve multipart treatments while preserving nulls**
   - Dissolve polygons by a list of treatment and admin fields (e.g., `TreatmentID`, `ProjectID`, `TreatmentName`, dates, acres, status).
   - In `dissolve_with_nulls`, temporarily replace nulls with placeholders (numeric, datetime, string), dissolve, then restore nulls and restore integer types where allowed.
6. **Attach Task Force standard columns**
   - Rename `ProjectID` → `PrjID`.
   - Call `add_common_columns` to add the Task Force standardized schema to the dissolved GeoDataFrame.
7. **Map NPS attributes into the standardized schema**
   - Set `PROJECTID_USER` to `PrjID` if present, otherwise `TreatmentID`.
   - Assign static fields:
     - `AGENCY = 'NPS'`
     - `ORG_ADMIN_p = 'NPS'`
     - `PROJECT_CONTACT = 'Kent van Wagtendonk'`

        - `PROJECT_EMAIL = 'Kent_Van_Wagtendonk@nps.gov'`
        - `PRIMARY_FUNDING_SOURCE = 'NPS'`
        - `PRIMARY_FUNDING_ORG = 'OTHER'`
        - `BVT_USERD = 'NO'`
        - `ACTIVITY_UOM = 'AC'`
        - `ADMIN_ORG_NAME = 'NPS'`
        - `PRIMARY_FUND_SRC_NAME = 'NPS'`
        - `PRIMARY_FUND_ORG_NAME = 'NPS'`
        - `Source = 'nps_flat_fuelstreatments'`
    - Map additional fields from NPS columns:
        - `ADMINISTERING_ORG = 'NPS'`
        - `PROJECT_NAME` and `PROJECTNAME_` from `TreatmentName`
        - `IMPLEMENTING_ORG` and `IMPLEM_ORG_NAME` from `UnitName`
        - `ORG_ADMIN_t = None`
        - `ACTIVITY_QUANTITY` from `TreatmentAcres`.
8. **Derive activity dates, status, and crosswalk**
    - Compute `ACTIVITY_END`:
        - If `ActualCompletionDate` exists, use it.
        - Else if `ActualCompletionFiscalYear` exists, use October 1 of that year.

        - Else leave as `None`.
    - Derive `ACTIVITY_STATUS` from `TreatmentStatus`:
        - `"Completed"` → `"COMPLETE"`
        - `"Initiated"` → `"ACTIVE"`
        - Otherwise → `"TBD"`.
    - Extract `Year` from `ActualCompletionDate`.
    - Compute `Crosswalk`:
        - If `TreatmentType` is present, use `TreatmentType`.
        - If `TreatmentType` is missing and `TreatmentCategory == "Fire"`, use `"Broadcast Burn"`.
        - If `TreatmentType` is missing and `TreatmentCategory == "Mechanical"`, use `"Hand Pile Burn"`.
        - Otherwise `None`.
9. **Filter to requested year range and trim fields**
    - Keep only records where `Year` is between `start_year` and `end_year` (inclusive).
    - Use `keep_fields` to drop columns that are not part of the standardized Task Force output.
10. **Enrich with reference layers**
    - Call `enrich_polygons(...)` on the filtered GeoDataFrame to add attributes such as vegetation, ownership, county, WUI, Task Force Region, and related fields (as implemented inside `enrich_polygons`).
11. **Construct standardized treatment ID**

- Build `TRMTID_USER` using a safe concatenation of:
  - First 7 characters of `PROJECTID_USER`
  - First 3 characters of `COUNTY`
  - First 4 characters of `PRIMARY_OWNERSHIP_GROUP`
  - First 3 characters of `IN_WUI`
  - First 8 characters of `PRIMARY_OBJECTIVE`
- Handle null/empty parts safely so rows without components get `None` for `TRMTID_USER`.

12. **Apply domains and NPS-specific final adjustments**
    - Call `assign_domains(...)` to apply domain/coded-value mappings to the enriched data.
    - Set `ADMINISTERING_ORG` from `ADMIN_ORG_NAME`, filling any remaining nulls with `'NPS'`.
    - Fill any missing `AGENCY` values with `'NPS'`.
13. **Apply MAS cutoff and save output**
    - For treatments with `ACTIVITY_END >= '2023-10-01'`, set `COUNTS_TO_MAS = 'NO'`.
    - Save the final enriched NPS layer to the output geodatabase with group name `"c_Enriched"`.
14. **Runtime diagnostics**
    - In `__main__`, after processing, compute and log the process memory usage in MB.

PFIRS

1. **Load PFIRS data and lookup table**
    - Read the PFIRS layer from the input geodatabase using `gpd.read_file(..., layer=pfirs_layer_name)`.
    - Read an Excel lookup table (`pfirs_lookup_df`) for agency remapping and anonymous timber flags.
2. **Build lookup structures**
    - Create `pfirs_lookup_dict` that maps original `Agency` names to standardized agency labels (`agency_calc_field_2`).
    - Build `anonymous_org_list` of agencies whose `"Org Type"` is `"Industrial Timber"`.
3. **Validate and reproject PFIRS data**
    - Use `verify_gdf_columns` to ensure PFIRS contains all required `PFIRS_COLUMNS`.
    - Reproject PFIRS geometries to `EPSG:3310` and log columns with `show_columns`.
4. **Filter out agencies that are handled by other pipelines**
    - Remove PFIRS records where `Agency` is in:
        - `Cal Fire`, `US Forest Service`, `US Fish and Wildlife Services`, `Bureau of Land Management`, `National Park Service`.
5. **Rename key fields and add standard Task Force columns**
    - Rename `Agency → AGENCY_` and `County → COUNTY_`.
    - Call `add_common_columns(gdf)` to attach the Task Force standardized column set.

6. **Populate standardized project and organization fields**
    - Set `PROJECTID_USER = 'PFIRS-' + index`.
    - Map `AGENCY` using the lookup dictionary (`pfirs_lookup_dict`), defaulting to `'OTHER'`.
    - Set:
        - `ORG_ADMIN_p = AGENCY`
        - `PROJECT_CONTACT = 'Jason Branz'`
        - `PROJECT_EMAIL = 'jason.branz@arb.ca.gov'`
        - `ADMINISTERING_ORG = AGENCY`
        - `PROJECT_NAME` from `Burn_Unit`
        - `PRIMARY_FUNDING_SOURCE = 'LOCAL'`
        - `PRIMARY_FUNDING_ORG = 'OTHER'`.
    - Set `IMPLEMENTING_ORG` to `'Timber Companies'` when `AGENCY_` is in `anonymous_org_list`, otherwise keep original `AGENCY_`.
    - Set `TRMTID_USER = 'PFIRS-' + index`.
    - Initialize `PROJECTNAME_ = None`, `ORG_ADMIN_t = None`, `BVT_USERD = 'NO'`.
7. **Populate activity-level fields**
    - Set:
        - `ACTIVITY_END` from `Burn_Date`
        - `ACTIVITY_STATUS = 'COMPLETE'`

        - `ACTIVITY_QUANTITY` from `Acres_Burned`
        - `ACTIVITY_UOM = 'AC'`
        - `ADMIN_ORG_NAME = 'CARB'`
        - `IMPLEM_ORG_NAME` from `AGENCY_`
        - `PRIMARY_FUND_SRC_NAME = 'LOCAL'`
        - `PRIMARY_FUND_ORG_NAME = 'OTHER'`
        - `Source = 'PIFIRS'` (note spelling in code).

8. **Map burn types into Crosswalk codes**
    - Use `map_burn_type` to convert `Burn_Type` into standardized descriptors, for example:
        - `'Broadcast'`, `'Unknown'`, `'Multiple Fuels'`, `'UNK'` → `'Broadcast Burn'`
        - `'Hand Pile'`, `'Pile'`, `'Mixed Piles'` → `'Hand Pile Burn'`
        - `'Machine Pile'` → `'Machine Pile Burn'`
        - `'Landing Pile'` → `'Landing Pile Burn'`
    - Store result in `Crosswalk`.

9. **Filter PFIRS records by agency and burn type**
    - Build:
        - `mask_1`: records from non-timber agencies and from `'FWS Forestry LLC'`.
        - `mask_2`: records from timber companies where `Burn_Type != 'Broadcast'`.

- Keep records where `mask_1` **or** `mask_2` is `True`, dropping broadcast burns from anonymous timber companies except `FWS Forestry LLC`.

10. **Reduce to standard output schema**
    - Call `keep_fields(gdf)` to keep only the standardized Task Force fields.
    - Log the remaining schema with `show_columns`.
11. **Enrich PFIRS points with reference data**
    - Call `enrich_points(gdf, a_reference_gdb_path, start_year, end_year)` to add spatial enrichment (as implemented inside `enrich_points`).
    - Store result in `enriched_gdf`.
12. **Assign coded domains**
    - Call `assign_domains(enriched_gdf)` to apply domain/coded-value mappings.
13. **Optionally remove duplicates using treatment polygons**
    - If `treat_poly_gdf` is provided:
        - Filter treatment polygons to those with `ACTIVITY_DESCRIPTION` in `['BROADCAST_BURN', 'PILE_BURN']` (`rx_burns`).
        - For each `Year_txt` in `enriched_gdf`:
            - Subset `rx_burns` and `enriched_gdf` to that year.
            - Run a spatial join (`gpd.sjoin`) to find PFIRS points that intersect treatment polygons.
            - Keep only PFIRS point indices **not** intersecting polygons (build `out_index`).
        - Subset `enriched_gdf` to `out_index` to drop PFIRS records that spatially duplicate existing treatment polygons.

14. **Finalize AGENCY and save output**
    - Overwrite `AGENCY` with `ORG_ADMIN_p` as a temporary fix.
    - Save `enriched_gdf` to the specified output geodatabase and layer via `save_gdf_to_gdb(..., group_name="c_Enriched")`.
15. **Main script orchestration**
    - In `__main__`:
        - Load paths, layer names, years, lookup Excel, and appended treatment polygons from `config.yaml`.
        - Read `treat_poly_gdf` from the appended treatment geodatabase.
        - Call `enrich_PFIRS(...)` with all inputs.
        - Log process memory usage in MB at the end.

TIMBER SPATIAL

1. **Load Timber Industry spatial data**
    - Read the timber industry layer from the input geodatabase (including `OBJECTID`).
    - If the `Organization` column is missing, copy it from `Org_Public`.
2. **Validate schema and prepare geometry**
    - Verify that the input data contains all required `TIMBER_INDUSTRY_SPATIAL_COLUMNS`.
    - Reproject geometries to EPSG:3310.
    - Normalize `Status` by replacing `"Exists"` with `"Active"`.

- Log the available columns.

3. **Clip features to California**
    - Use `clip_to_california` with the reference geodatabase to keep only polygons within California.
4. **Repair geometries**
    - Call `repair_geometries` to fix invalid geometries in the clipped dataset.
5. **Add Task Force common columns**
    - Apply `add_common_columns` to create the standardized Task Force schema on the clipped data.
6. **Transfer and derive project- and org-level attributes**
    - Set `PROJECTID_USER` to `'TI-' + OBJECTID`.
    - Set:
        - `AGENCY = "TIMBER"`
        - `IMPLEMENTING_ORG` from `Organization`
        - `ORG_ADMIN_p`, `ORG_ADMIN_t`, `ORG_ADMIN_a` to `"TIMBER"`
        - `PROJECT_CONTACT` and `PROJECT_EMAIL` to `None`
        - `ADMINISTERING_ORG = "TIMBER"`
        - `PROJECT_NAME` from `Name`
        - `PRIMARY_FUNDING_SOURCE = "PRIVATE"`
        - `PRIMARY_FUNDING_ORG = "TIMBER"`
        - `ACTIVITY_NAME` from `Name`

        - `BVT_USERD = "NO"`.

7. **Calculate activity start and end dates from year**
    - Convert `Year` to numeric.
    - Set `ACTIVITY_START` to January 1 of the `Year`.
    - Set `ACTIVITY_END` to December 31 of the `Year` (both via `pd.to_datetime`).
8. **Set activity and project status**
    - Set `ACTIVITY_STATUS` to the uppercase of `Status`.
    - Set `PROJECT_STATUS` to the uppercase of `Status`.
9. **Set activity quantity and units**
    - Set `ACTIVITY_QUANTITY` from `GISACRES` (cast to float).
    - Set `ACTIVITY_UOM = "AC"`.
10. **Fill admin and funding name fields**
    - Set:
        - `ADMIN_ORG_NAME = "TIMBER"`
        - `IMPLEM_ORG_NAME` from `Organization`
        - `PRIMARY_FUND_SRC_NAME` from `PRIMARY_FUNDING_SOURCE`
        - `PRIMARY_FUND_ORG_NAME` from `PRIMARY_FUNDING_ORG`
        - `Source = "Industrial Timber"`.
11. **Populate activity description and crosswalk**
    - Set `ACTIVITY_DESCRIPTION` from `Objective`.

- Set `Crosswalk` equal to `ACTIVITY_DESCRIPTION`.

12. **Filter by reporting years**
    - Keep only records where `Year` lies between `start_year` and `end_year` (inclusive), creating `selected_gdf`.

13. **Load Task Force template and append data**
    - Create an empty GeoDataFrame with columns from `get_wfr_tf_template(...)` and CRS = EPSG:3310.
    - Append `selected_gdf` to this template.
    - Set `TRMT_GEOM = "POLYGON"`.

14. **Reduce to standard output fields**
    - Call `keep_fields` to retain only the standard Task Force fields.

15. **Enrich polygons with reference layers**
    - Run `enrich_polygons` to add vegetation, ownership, county, WUI, region, and related attributes based on the reference geodatabase and year range.

16. **Assign treatment ID**
    - Set `TRMTID_USER` equal to `PROJECTID_USER`.

17. **Apply domains/coded values and save output**
    - Call `assign_domains` to map fields to their domain/coded values.
    - Save the enriched GeoDataFrame to the specified output geodatabase and layer via `save_gdf_to_gdb(..., group_name="c_Enriched")`.

18. **Main script orchestration**
    - In `__main__`, read configuration from `config.yaml`, build input/output paths and year range, call `enrich_Timber_Industry(...)`, and log final memory usage.

TIMBER NON-SPATIAL

1. **Load nonspatial Excel data**
    - Open the Excel file and read the first sheet into a `DataFrame` (`tn_df`).
    - Validate that all required `TIMBER_NONSPATIAL_COLUMNS` are present.
2. **Rename source columns to internal working fields**
    - Rename human-readable columns (e.g., `"ACTIVITY CATEGORY"`, `"BROAD VEGETATION TYPE"`, `"ADMINISTERING ORG"`, `"COUNTY"`, `"IN WUI"`, etc.) to internal names with underscores (e.g., `ACTIVITY_CATEGORY`, `BROAD_VEGETATION_TYPE_`, `ADMINISTERING_ORG_`, `COUNTY_`, `IN_WUI_`, etc.).
3. **Add Task Force common columns**
    - Apply `add_common_columns(tn_df)` to introduce the standard Task Force schema fields onto this table.
4. **Populate standardized fields from the renamed input columns**
    - Copy values from the `_` suffixed working fields into Task Force standard fields, for example:
        - `ACTIVITY_DESCRIPTION` ← `ACTIVITY_DESCRIPTION_`
        - `ACTIVITY_CAT` ← `ACTIVITY_CATEGORY`
        - `BROAD_VEGETATION_TYPE` ← `BROAD_VEGETATION_TYPE_`
        - `BVT_USERD` ← `IS_BVT_USER_DEFINED`
        - `ACTIVITY_STATUS` ← `ACTIVITY_STATUS_`
        - `ACTIVITY_QUANTITY` ← `ACTIVITY_QUANTITY_`
        - `ACTIVITY_UOM` ← `ACTIVITY_UNITS`

        - `ACTIVITY_START` / `ACTIVITY_END` from their `_` versions
        - `ADMIN_ORG_NAME`, `COUNTY`, `IN_WUI`, `REGION`, `PRIMARY_OWNERSHIP_GROUP` from corresponding `_` columns.
    - Strip whitespace from `ACTIVITY_DESCRIPTION`.
5. **Set funding, agency, and organizational fields**
    - Set constant/derived values:
        - `PRIMARY_FUNDING_SOURCE = 'PRIVATE'`
        - `PRIMARY_FUNDING_ORG = 'PRIVATE_INDUSTRY'`
        - `AGENCY`, `ADMINISTERING_ORG`, `IMPLEMENTING_ORG`, `ORG_ADMIN_p`, `ORG_ADMIN_t`, `ORG_ADMIN_a` all from `ADMIN_ORG_NAME`
        - `PRIMARY_FUND_SRC_NAME` and `PRIMARY_FUND_ORG_NAME` from the funding fields
        - `Source = 'Industrial Timber'`
        - `TRMT_GEOM = 'POINT'`.
6. **Normalize some activity descriptions and set Crosswalk**
    - Replace specific activity labels (e.g., `"Fuel Break (pursuant to FPRs)" → "Thinning (Manual)"`, `"Group Selection" → "Group Selection Harvest"`, etc.).
    - Initialize `Crosswalk = ACTIVITY_DESCRIPTION`.
7. **Define reference coordinates per activity type and a grid of fake locations**
    - Create `lat_mapping` and `lon_mapping` dictionaries keyed by `ACTIVITY_DESCRIPTION`.
    - Compute min/max lat/lon from those dictionaries.

- Build a regular grid of coordinates over that bounding box using `np.mgrid`.
- Sort `tn_df` by `ACTIVITY_DESCRIPTION`.
- Assign the generated grid coordinates to `tn_df.geometry` as `Point` objects (this creates an internal geometry arrangement for the records).

8. **Convert table to a GeoDataFrame with lat/long**
   - Create a new geometry list via `Point(xy)` for each `(LONGITUDE, LATITUDE)` pair.
   - Construct a GeoDataFrame `gdf` with CRS `EPSG:4326`, then reproject it to `EPSG:3310`.
9. **Dissolve points by key attributes and sum quantities**
   - Define `essential_fields` (e.g., `ACTIVITY_DESCRIPTION`, `ACTIVITY_STATUS`, `COUNTY`, `BROAD_VEGETATION_TYPE`, `ACTIVITY_START`, `ACTIVITY_END`, `ADMIN_ORG_NAME`).
   - Fill `NaN` in these essential fields with empty strings.
   - Build an aggregation dictionary:
     - `ACTIVITY_QUANTITY` is summed.
     - Most other fields (excluding geometry and essential fields) use `'first'`.
   - Dissolve `gdf` by `essential_fields` using this aggregation, producing `gdf_dissolved`.
10. **Generate project and treatment IDs**
    - Set:
      - `PROJECTID_USER = 'TI-' + index`
      - `PROJECT_NAME = PROJECTID_USER`

        - `TRMTID_USER = PROJECTID_USER`
        - `PROJECTNAME_ = None`.

11. **Reduce to the standard Task Force fields**
    - Call `keep_fields(gdf_dissolved)` to drop non-standard columns and retain only the Task Force schema.
    - Log final columns and shape.

12. **Enrich point features with reference datasets**
    - Call `enrich_points(...)` using the reference geodatabase and year bounds to add vegetation, ownership, county, WUI, region, etc., returning `tn_enriched`.

13. **Apply post-enrichment corrections (`update_enriched_data`)**
    - For specific `ACTIVITY_DESCRIPTION` values (e.g., `"CHIPPING"`, `"LOP_AND_SCAT"`, `"MASTICATION"`, `"TREE_PLNTING"`, etc.), set:
        - `ACTIVITY_CAT` and `PRIMARY_OBJECTIVE` from `activity_mapping`.
    - For certain `Crosswalk` values (e.g., `"Roadway Clearance"`, `"Sanitation Harvest"`, `"Landing Treated"`, `"Invasive Plant Removal"`, etc.), update:
        - `ACTIVITY_DESCRIPTION`, `ACTIVITY_CAT`, and `PRIMARY_OBJECTIVE` using `value_mapping`.
    - Set `Crosswalk = ACTIVITY_DESCRIPTION`.
    - Set `COUNTS_TO_MAS = 'YES'`.
    - Set `ADMINISTERING_ORG = AGENCY`.

14. **Assign domains and coded values**
    - Call `assign_domains(tn_enriched)` to map fields to domain/coded values.

15. **Save enriched nonspatial timber treatments**

- Write the final enriched GeoDataFrame to the output geodatabase and layer via `save_gdf_to_gdb(..., group_name="c_Enriched")`.

16. **Main script orchestration**

    - In `__main__`, read configuration from `config.yaml`, derive input/output paths and year range, call `enrich_Timber_Nonspatial(...)`, and log the process memory usage.

USFS

1. **Load USFS FACTS data with caching**

    - Build a cache file name from the input GDB and layer name.
    - If a cached Parquet file exists, load the GeoDataFrame from `cache/...parquet`.
    - Otherwise, read the layer from the input GDB with `gpd.read_file(...)` (including `OBJECTID`) and save it to Parquet for future runs.

2. **Validate schema and set projection**

    - Check that all required `USFS_COLUMNS` are present using `verify_gdf_columns`.
    - Reproject the dataset to EPSG:3310.
    - Log the available columns via `show_columns`.

3. **Remove records with missing geometry and filter to California**

    - Count and drop all rows where `geometry` is `NaN`.
    - Keep only rows where `STATE_ABBR == 'CA'`.

4. **Select activity records of interest**

    - Filter to rows whose `ACTIVITY_CODE` is in a specified list of codes.

- Further refine:
    - Keep all rows except those with activity codes `1117`, `1119`, `2510`, `2341` in a base subset.
    - For `1117` and `1119`, keep only records where `FUELS_KEYPOINT_AREA == '6'`.
    - For `2510` and `2341`, keep only records where `FUELS_KEYPOINT_AREA` is `'6'` or `'3'`.
- Concatenate these subsets into a single GeoDataFrame.

5. **Filter by date availability and completion year**
    - Keep only records where at least one of `DATE_COMPLETED`, `DATE_AWARDED`, or `NEPA_SIGNED_DATE` is non-null.
    - Define a start date of `start_year-01-01` (UTC).
    - Keep only records where `DATE_COMPLETED` is on or after this start date (or treated as max timestamp if null), enforcing a post-1995 filter via the given `start_year`.
6. **Repair geometry**
    - Run `repair_geometries` on the filtered GeoDataFrame to fix invalid geometries.
7. **Rename fields and add Task Force common columns**
    - Rename:
        - `TREATMENT_NAME → TREATMENT_NAME_FACTS`
        - `LATITUDE → LATITUDE_`
        - `LONGITUDE → LONGITUDE_`
    - Call `add_common_columns` to attach the standard Task Force schema fields.

8. **Populate core ID and organizational fields**
    - Set:
        - `PROJECTID_USER = 'USFS-' + NEPA_DOC_NBR`
        - `AGENCY = 'USDA'`
        - `ORG_ADMIN_p`, `ORG_ADMIN_t`, `ORG_ADMIN_a = 'USFS'`
        - `PROJECT_CONTACT = 'Tawndria Melville'`
        - `PROJECT_EMAIL = 'tawndria.melville@usda.gov'`
        - `ADMINISTERING_ORG = 'USFS'`
        - `PRIMARY_FUNDING_SOURCE = 'FEDERAL'`
        - `PRIMARY_FUNDING_ORG = 'USFS'`
        - `IMPLEMENTING_ORG = 'Pacific Southwest Regional Office'`
        - `TRMTID_USER = SUID`
        - `ACTIVID_USER = SUID-OBJECTID`
        - `BVT_USERD = 'NO'`
    - Convert `ISWUI` values `Y`/`N` into `IN_WUI = 'WUI_USER_DEFINED'` or `'NON-WUI_USER_DEFINED'`.
9. **Compute activity end date**
    - Set `ACTIVITY_END = DATE_COMPLETED`.
    - For rows where `ACTIVITY_END` is null, replace it with `NEPA_SIGNED_DATE`.
10. **Derive activity status**

- Use a custom `get_status` function:
  - `COMPLETE` if `DATE_COMPLETED` is not null.
  - `ACTIVE` if not complete but `DATE_AWARDED` is not null.
  - `OUTYEAR` if neither is set but `NEPA_SIGNED_DATE ≥ 2025-01-24` (UTC).
  - `PLANNED` if `NEPA_SIGNED_DATE ≥ 2014-01-24` (UTC) and earlier than the OUTYEAR cutoff.
  - `CANCELLED` otherwise.
- Store the result in `ACTIVITY_STATUS`.

11. **Set activity quantity and units**
    - `ACTIVITY_QUANTITY = NBR_UNITS_ACCOMPLISHED` if present, otherwise `NBR_UNITS_PLANNED`.
    - `ACTIVITY_UOM = UOM`.
12. **Assign admin, funding, and naming fields**
    - Set:
      - `ADMIN_ORG_NAME = 'USFS'`
      - `IMPLEM_ORG_NAME = WORKFORCE_CODE`
      - `PRIMARY_FUND_SRC_NAME = 'USFS'`
      - `PRIMARY_FUND_ORG_NAME = 'USFS'`
      - `ACTIVITY_NAME = None`
      - `Source = 'usfs_treatments'`
      - `Year = ACTIVITY_END.year`

- Normalize one specific activity label:
  - If `ACTIVITY == 'Piling of Fuels, Hand or Machine '`, set `Crosswalk = 'Piling of Fuels, Hand or Machine'`; otherwise `Crosswalk = ACTIVITY`.

13. **Set treatment geometry type and activity code integer**
    - Define `TRMT_GEOM` based on the `ACTIVITY` value:
      - `'A'` → `'POLYGON'`
      - `'L'` → `'LINE'`
      - `'P'` → `'POINT'`
      - Anything else → `'POLYGON'`.
    - Create `Act_Code` as an integer version of `ACTIVITY_CODE`.
14. **Reduce to Task Force standard fields**
    - Call `keep_fields` to drop non-standard columns and keep only fields required by the Task Force schema.
15. **Filter to the requested year range**
    - Keep only rows where `Year` is between `start_year` and `end_year` (inclusive).
16. **Enrich geometries with Task Force spatial attributes**
    - Call `enrich_polygons(gdf_filtered, a_reference_gdb_path, start_year, end_year, manager=manager)` to attach Task Force enrichment fields (using the provided reference GDB and optional multiprocessing manager).
17. **Assign domain / coded values**
    - Call `assign_domains` on the enriched GeoDataFrame to apply domain mappings and coded value lookups.

18. **Save the enriched USFS treatments**
    - Write the final `enriched_gdf` to the output file geodatabase and layer using `save_gdf_to_gdb(..., group_name="c_Enriched")`.
19. **Main script: process all configured USFS regions**
    - Read configuration from `config.yaml` to get:
        - USFS input base path, GDB name template, layer name, reference GDB, year range, and output GDB path template.
    - For each configured `region_id`:
        - Build the region-specific input GDB path and output layer name.
        - Call `enrich_USFS` with the corresponding paths and parameters.
    - After processing all regions, log the process memory usage in MB.

Append.py

1. **Discover all enriched GDBs and classify layers by geometry**
    - List all files in the configured `enriched_path`.
    - Skip `appended.gdb` and `reports.gdb`.
    - For each remaining GDB:
        - Use `gpd.list_layers` to inspect all layers.
        - For each layer, classify it as:
            - `point` if its `geometry_type` contains `"point"`,
            - `line` if it contains `"line"`,

        - otherwise `polygon`.
    - Store each layer as a dict: `{'gdb_path': <path>, 'layer_name': <name>}`.
- Return a dict of lists: `{'point': [...], 'line': [...], 'polygon': [...]}`.

2. **Optionally filter which geometry types to process (CLI)**
    - Read `--geom_type` from command line (default `"all"`).
    - If `geom_type` is `"point"`, `"line"`, or `"polygon"`, keep only that geometry type's layers and empty the others.
    - If `"all"`, keep everything.
    - Raise a `ValueError` for any other input.
3. **Force domain/categorization standardization on all selected layers**
    - For each geometry type (`point`, `line`, `polygon`) and each layer path dict:
        - Read the layer from its GDB using `gpd.read_file`.
        - Pipe the GeoDataFrame through:
            - `categorize_activity(gdf)`
            - `standardize_domains(...)`
            - `counts_to_mas(..., start_year, end_year)`
        - Overwrite the layer back into the same GDB with the standardized result.
4. **Load and append enriched polygon/line/point features**
    - For each geometry collection (`polygon`, `line`, `point`), call `append_enriched_features`:

Inside **`append_enriched_features`**:

- Initialize an empty list `gdfs_to_append`.
- For each layer dict:
  - Read the layer into a GeoDataFrame.
  - If its CRS is not `"EPSG:3310"`, reproject to `"EPSG:3310"`.
  - Use `get_rows_with_empty_geometry` to detect rows with empty geometry.
    - If empty geometries exist:
      - Subset those rows as `gdf_na`.
      - If all of them have `COUNTS_TO_MAS == 'NO'`, drop those rows from `gdf`.
      - Otherwise, log an error message and call `exit()`.
  - Append the cleaned `gdf` to `gdfs_to_append`.
- If `gdfs_to_append` is non-empty, concatenate them with `pd.concat(..., ignore_index=True)` to form `final_gdf`.
- If empty, set `final_gdf = None`.
- Return `final_gdf`.

- Capture the three concatenated results as `enriched_polygons`, `enriched_lines`, and `enriched_points`.

5. **Load California boundary for clipping**
   - Read the California boundary layer from the reference GDB using the configured `california_boundary_layer_name`.
6. **Identify the Timber Nonspatial enriched layer (if present)**

- Initialize `timber_nonspatial_path = None`.
- Iterate over all `point` layers in `enriched_layers['point']`:
    - If the `gdb_path` string contains `'Timber_Nonspatial'`, store that dict as `timber_nonspatial_path` and stop searching.
- If `timber_nonspatial_path` was found:
    - Read that layer into a GeoDataFrame `timber_nonspatial`.

7. **Clip all appended geometries to California using Dask GeoPandas**
    - For each of the three concatenated GeoDataFrames and output layer names:
        - (`enriched_polygons`, `"appended_poly"`),
          (`enriched_lines`, `"appended_line"`),
          (`enriched_points`, `"appended_point"`):
            - Convert the GeoDataFrame to a Dask GeoDataFrame with 16 partitions using `dask_geopandas.from_geopandas`.
            - Perform a spatial join with the California boundary:
                - `ddf.sjoin(california_boundary, how='inner', predicate='intersects').compute()`
                - This keeps only features that intersect the California boundary.
            - Drop the boundary artifact columns: `index_right`, `Shape_Area`, `Shape_Length`.
            - If the current output layer name is `"appended_point"`:
                - Concatenate the clipped points with the `timber_nonspatial` GeoDataFrame so that Timber nonspatial points (which are outside California bounds and would otherwise be clipped out) are added back.
            - Save the resulting clipped GeoDataFrame to the `output_append_path` GDB with the current layer name using

`save_gdf_to_gdb`.

8. **Configuration and setup in `__main__`**
    - Load `config.yaml` and read:
        - `enriched_path` for input enriched GDBs.
        - `reference_gdb` and `california_boundry_layer_name` for the clipping boundary.
        - `output_append_path` for the appended output GDB.
        - `start_year` and `end_year` for `counts_to_mas`.
    - Call `get_all_enriched_paths(enriched_path)` to discover and classify all enriched layers.
    - Apply the geometry-type filter from the `--geom_type` argument.
    - Reapply domain/categorization logic to all selected layers.
    - Build the concatenated, clipped, and saved appended layers as described above.

Activity_Report.py

1. **Load configuration and set paths**
    - Read `config.yaml`.
    - Get:
        - `append_path`: path to the appended geodatabase (with enriched point/line/polygon layers).
        - `output_report_path`: path for the activity report geodatabase.

        - `start_year` and `end_year` (not used in the current code logic).
    - Build a dated `report_layer_name` using today's date.
2. **Read appended enriched layers**
    - Read three layers from the appended GDB:
        - `appended_poly` into `enriched_polygons`.
        - `appended_point` into `enriched_points`.
        - `appended_line` into `enriched_lines`.
3. **Combine enriched features and filter to MAS-counting activities**
    - Concatenate lines, points, and polygons into a single GeoDataFrame `append_all` using `pd.concat`.
    - Keep only rows where `COUNTS_TO_MAS == 'YES'`.
4. **Rebuild geometry as points from coordinates**
    - Replace `append_all.geometry` with point geometries created from the `LONGITUDE` and `LATITUDE` columns using `gpd.points_from_xy`.
5. **Select report fields**
    - Subset `append_all` to the following columns:
        - `"AGENCY"`, `"ADMINISTERING_ORG"`, `"PRIMARY_OWNERSHIP_GROUP"`, `"COUNTY"`, `"REGION"`, `"ACTIVITY_DESCRIPTION"`, `"ACTIVITY_CAT"`, `"BROAD_VEGETATION_TYPE"`, `"ACTIVITY_STATUS"`, `"ACTIVITY_QUANTITY"`, `"ACTIVITY_UOM"`, `"ACTIVITY_END"`, `"Year_txt"`, and `"geometry"`.
6. **Filter to valid geometries**
    - Keep only rows where the geometry is valid (`append_all.is_valid`).

7. **Derive ENTITY_TYPE from AGENCY**
    - Define `get_entity_type(agency)` that maps:
        - State: `['CALEPA', 'CALSTA', 'CNRA', 'PARKS', 'California State Parks']`
        - Federal: `['DOD', 'DOI', 'USDA', 'DOE', 'NPS']`
        - Timber Companies: `['Industrial Timber', 'Timber Companies', 'TIMBER']`
        - All other agencies → `None`.
    - Add `ENTITY_TYPE` by applying this function to the `AGENCY` column.
8. **Save the activity report**
    - Call `get_activity_report(...)` with enriched points, lines, and polygons to create `activity_report_gdf`.
    - Save `activity_report_gdf` to `output_report_path` with `report_layer_name` using `save_gdf_to_gdb`.
    -

Here's a concise, code-accurate, step-by-step description of what this script does.

1. **Read configuration and inputs (main block)**
    - Load `config.yaml`.
    - Get paths and parameters:
        - `input_append_path` (appended enriched GDB) and layer names for points, lines, polygons.
        - `output_report_path` for footprint report GDB.
        - `veg_path` and `veg_layer_name` (vegetation/ownership GDB and layer).

        - `temp_path` for intermediate outputs.
        - `start_year` and `end_year` for the footprint analysis.
    - Call `get_footprint(...)` with these settings.
    - Log memory usage at the end.
2. **Load base datasets in `get_footprint`**
    - Read vegetation/ownership GeoDataFrame (`veg_gdf`) from `veg_path` / `veg_layer_name`.
    - Read enriched Task Force layers from the appended GDB:
        - `enriched_points`
        - `enriched_lines`
        - `enriched_polygons`
    - Define `enrich_cols = ['AGENCY', 'IN_WUI', 'PRIMARY_OWNERSHIP_GROUP', 'BROAD_VEGETATION_TYPE', 'REGION', 'COUNTY']` for later reattachment after dissolves.
3. **Buffer / filter enriched geometries to represent activity area**
    - **Points (`update_pt`)**
        - Initialize `BufferMeters` column as `None`.
        - For records with non-null `ACTIVITY_QUANTITY` and `ACTIVITY_UOM == 'AC'`, compute a circular buffer radius (meters) from reported acres.
        - Keep only rows with `COUNTS_TO_MAS == 'YES'` and valid `BufferMeters > 0`.
        - Replace point geometry with circular buffers of size `BufferMeters`.
    - **Lines (`update_ln`)**

        - Initialize `BufferMeters` column as `None`.
        - Compute line lengths.
        - Identify lines with:
            - Non-null `ACTIVITY_QUANTITY`,
            - `ACTIVITY_UOM` equal to `'AC'` (case-insensitive),
            - Length > 0.
        - For those, compute `BufferMeters` so that a buffered line polygons has the correct area (in acres).
        - Keep lines with `COUNTS_TO_MAS == 'YES'` and non-null `BufferMeters`, excluding cases where `BufferMeters >= 200` and `Source == 'CalTrans'`.
        - For CalTrans records, call `get_max_treatment(...)` and merge back a reduced set of records.
        - Replace line geometry with flat-cap buffers of width `BufferMeters`.
    - **Polygons (`update_poly`)**
        - Filter polygons to those with `COUNTS_TO_MAS == 'YES'` and `TREATMENT_AREA < 100000`.

    The results are `buffered_pt`, `buffered_ln`, and `buffered_poly`.

4. **Loop over years and build per-year intermediate layers in `get_footprint`**
   For each year `y` from `start_year` to `end_year` (converted to string):

   4.1 **Filter buffered geometries by year**

    - `poly_cur_y`: polygons where `Year_txt == y`.
    - `ln_cur_y`: lines where `Year_txt == y`.

- `pt_cur_y`: points where `Year_txt == y`.

4.2 **Separate Timber nonspatial activities**

- `timber_cur_y`: subset of `pt_cur_y` with `AGENCY == 'TIMBER'`.
- These are excluded from subsequent spatial operations and appended later.

4.3 **Combine polygons and non-timber points**

- Concatenate `poly_cur_y` with non-timber points (`pt_cur_y[pt_cur_y.AGENCY != 'TIMBER']`) into `poly_pt`.
- Dissolve `poly_pt` (all features together) to get a single union geometry: `poly_pt_union_geom`.

4.4 **Split lines into intersecting and non-intersecting with polygon/point union**

- For `ln_cur_y`, compute `intersect_mask = ln_cur_y.intersects(poly_pt_union_geom)`.
- `ln_noi`: lines that do *not* intersect the union (`~intersect_mask`).
- `ln_intersect`: lines that *do* intersect the union (`intersect_mask`).

4.5 **Dissolve non-intersecting lines by treatment and reattach attributes**

- Dissolve `ln_noi` by `TRMTID_USER`, aggregating `ACTIVITY_QUANTITY` and `geometry` with `max`.
- Use `enrich_dissolved_columns(enrich_cols, ln_noi, dissolved_ln_noi)` to copy selected attribute columns (`AGENCY`, `IN_WUI`, etc.) from the original records to the dissolved result.

4.6 **Dissolve intersecting lines and compute “footprint” portion**

- Dissolve `ln_intersect` by `TRMTID_USER` in the same way.
- Reattach enrichment columns via `enrich_dissolved_columns(enrich_cols, ln_intersect, dissolved_ln_int)`.
- Compute a reduced `ACTIVITY_QUANTITY` for intersecting lines based on the portion outside the polygon/point union:
  - `footprint_ln_int = dissolved_ln_int.ACTIVITY_QUANTITY * (dissolved_ln_int.difference(poly_pt_union_geom).area / dissolved_ln_int.area)`
- Replace `dissolved_ln_int['ACTIVITY_QUANTITY']` with this footprint value.

4.7 **Dissolve polygon + point footprint by treatment and enrich attributes**

- Dissolve `poly_pt` by `TRMTID_USER`, aggregating `ACTIVITY_QUANTITY` and `geometry` with `max` to create `dissolved_cur_y`.
- Call `enrich_dissolved_columns(enrich_cols, poly_pt, dissolved_cur_y)` to attach enrichment fields.
- Reset index on `dissolved_cur_y`.
- Make geometries valid (`make_valid()`), which may convert invalid polygons to multipolygons.
- Remove duplicate geometries with `remove_duplicates`, which normalizes geometry and drops repeated shapes.

4.8 **Create representative points and write per-year temporary outputs**

- Copy `dissolved_cur_y` to `dissolved_point` and replace geometry with `representative_point()` of each polygon.
- Save per-year outputs to `temp_path` using `save_gdf_to_gdb`:
  - `original{y}`: full dissolved polygons with attributes.
  - `spaghetti{y}`: exploded polygon geometries (`dissolved_cur_y[['geometry']].explode()`).
  - `meatball{y}`: representative points.
  - `timber{y}`: concatenation of `timber_cur_y` and the two dissolved line GeoDataFrames (`dissolved_ln_int` and `dissolved_ln_noi`, after `reset_index()`).

5. **Create non-overlapping polygons with ArcPy in `get_nop_arcpy`**
   - For each year in the same range:
     - Build the output feature class name `Spaghetti_FeatureToPolygon{y}` inside `temp_path`.
     - Run `arcpy.management.FeatureToPolygon` on `spaghetti{y}` to create non-overlapping polygons.
     - Log the feature count for the result.

6. **Build final footprint outputs in `get_footprint_p2`**
   For each year:

   6.1 **Read per-year intermediate layers**
   - Read from `temp_path`:
     - `spaghetti` = `Spaghetti_FeatureToPolygon{y}`
     - `dissolved_point` = `meatball{y}`
     - `dissolved_cur_y` = `original{y}`

        - `timber_cur_y` = `timber{y}`

6.2 **Assign each non-overlapping polygon to a treatment and keep max area**

- Spatial join: `footprint_temp = spaghetti.sjoin(dissolved_point, how='left', predicate='intersects')`.
- Drop rows with missing join data, sort by `ACTIVITY_QUANTITY`, and for each `TRMTID_USER` keep the record with the largest `ACTIVITY_QUANTITY`.

6.3 **Recover original dissolved geometry for footprint treatments**

- Use `TRMTID_USER` in `footprint_temp` to select corresponding rows from `dissolved_cur_y` (via index on `TRMTID_USER`) to form `footprint_gdf`.

6.4 **Append timber/line footprints and save per-year report layer**

- Concatenate `footprint_gdf` with `timber_cur_y[footprint_gdf.columns]` into `footprint_enriched`.
- Save `footprint_enriched` to `report_path` as layer `Footprint_Report_{y}` using `save_gdf_to_gdb`.

7. **Execution**
    - `get_footprint(...)` orchestrates:
        - buffering and filtering points/lines/polygons,
        - writing year-specific temporary layers,
        - calling `get_nop_arcpy(...)` to build non-overlapping polygons, and
        - calling `get_footprint_p2(...)` to generate final per-year footprint report layers.