Wildfire Simulation Project Workflow and Design Explanation

Components of the Simulation

The project is organized into multiple modules/scripts, each responsible for different aspects of the simulation:

- **Environmental Factors Module** (final_environmental_factors.py): Handles the generation of environmental grids such as elevation, vegetation flammability, humidity, wind, and temperature for a given scenario. It uses historical climate data and geospatial data to create realistic initial conditions.
- **Historical Fire Data Analysis** (fire_grid_analysis.py): Processes past wildfire occurrence records to identify patterns (e.g. cells frequently burning). It produces an ignition_points grid highlighting cells that experienced fire in at least half of the years on record, which serves as likely ignition sources for the simulation.
- **Wildfire Simulation Module** (wildfire_simulation.py): Contains the core WildfireSimulation class and functions to run the fire spread simulation. It brings together the environmental grids and ignition points, simulates the fire spread over time, and allows multiple runs for statistical analysis.

These components work in concert as part of the overall workflow, described next.

Workflow Step 1: Environmental Data Preparation

(a) Climate and Weather Grids: Using the **EnvironmentalFactors** class, the simulation first prepares spatial grids of temperature, humidity, and wind for a randomly selected seasonal scenario. The code chooses one of four possible quarters (Q1, Q2, Q3, Q4) at random to represent a time of year. This "random quadrant" selection ensures the simulation isn't always starting in the same season – introducing variability in conditions like a different climate scenario for each run. Given the chosen quarter, the module reads a climate dataset (climate.csv) and filters data for that quarter to obtain distributions of daily temperature, relative humidity, and wind for that season. For example, to create the humidity grid it fits a normal distribution to historical relative humidity values for the chosen quarter and then draws random samples across the 25x25 grid. This produces a spatial humidity field with realistic variability, bounded between 0% and 100%. Similarly, it generates:

- a **temperature grid** by fitting a normal distribution to historical daily average temperatures for that quarter and sampling the grid (in °C),
- a **wind speed and direction grid** by fitting distributions to historical wind speeds and directions, then sampling a full grid (wind direction values are wrapped 0–360°),
- a **humidity grid** as mentioned, from historical relative humidity data for that season.

(b) Vegetation and Elevation Grids: The environmental module also provides static geographic factors:

- **Elevation grid:** fetched via an API (e.g. OpenTopography) or cached, giving the elevation (terrain height) at each grid cell. Elevation can affect fire spread (e.g. slope).

- **Vegetation ignition probability grid:** computed from land cover data. Multiple vegetation type maps (provided as CSVs) are combined – each cell gets an ignition likelihood based on the types of vegetation present and their flammability. For instance, grasslands may have a higher base ignition probability than urban areas. The code assigns baseline ignition probabilities to each vegetation class (e.g. dense shrub, grass, forest) and aggregates them onto the grid. If a special hypothesis scenario is invoked (like dense_shrub), the code increases the contribution of dense shrub vegetation (e.g. multiplying by 1.5) to simulate a heavier fuel load in shrubs, then scales the whole grid by a density factor. The result is a 0–1 map of how easily each cell can ignite due to vegetation fuel.

By the end of this preparation step, we have a set of grids ready: **Elevation**, **Vegetation Flammability**, **Temperature**, **Humidity**, **Wind Speed**, and **Wind Direction**, all aligned on the same grid. These serve as inputs to the fire spread simulation.

Workflow Step 2: Historical Fire Frequency Integration

Before the simulation starts, the historical fire data is used to determine **ignition points**. The fire_grid_analysis.pyscript reads wildfire occurrence records (e.g. satellite fire detections) over many years for the region and maps them onto the 25x25 grid. It then identifies cells that have burned repeatedly across years. Specifically, it computes how many years each cell experienced a fire and flags cells that had fire in at least half of the years observed. Those cells are marked in an **ignition frequency grid** (value 1 for frequent-fire cells, 0 otherwise) and saved as fire_frequent_cells_25x25_southeast_us.npy.
This step captures the idea that certain locations (due to lightning, human activity, or terrain) are more prone to ignition. Rather than starting fires randomly, the simulation will leverage this map to initiate fires in credible locations. In the **WildfireSimulation** initialization, this ignition grid is loaded and used as the starting fire map: cells marked as frequent ignition sources will be set on fire at time zero.
Why: Incorporating historical ignition patterns adds scientific realism – it reflects real-world conditions where fires often start in known hotspots (for example, dry lightning strike areas or areas of frequent human-caused fires). It ensures the simulation's starting conditions are not arbitrary, but grounded in data.

Workflow Step 3: Initializing the Wildfire Simulation
With environment and ignition data prepared, the simulation can begin. When a WildfireSimulation object is created (in wildfire_simulation.py), it performs the following initialization:

- **Grid Setup:** It defines a grid_size (default 25) and prepares a fire_grid matrix of that size, initially all zeros (no fire). It also sets a max_time_steps (default 24, representing the number of simulation steps -hours) and a small convergence_threshold (e.g. 0.01) for later use in stopping criteria.
- **Environmental Grids Loading:** The simulation instantiates the EnvironmentalFactors module for the given grid size, and immediately uses it to

generate or load the static grids: elevation and vegetation ignition probability. These remain constant throughout the simulation. It then selects a random quarter (season) and retrieves the climate condition grids for that quarter: humidity, wind speed & direction, and temperature. At this point, the simulation has all necessary environmental inputs for the first time step.

- **Initial Fire Ignition:** The fire_grid is seeded with the ignition points derived from historical data. In practice, this means any cell that was marked as a frequent igniter is set to a burning state. (In the code, the ignition_points array is loaded and those positions are copied into fire_grid as 1's to indicate active fire.) The number of initially burning cells is recorded as the first entry in a burned_areas list. From here, the fire can start to spread to adjacent cells in subsequent steps.

If a specific hypothesis mode is active, additional adjustments are made at initialization. For example, if the simulation is testing a **"high humidity"** hypothesis, after generating humidity it ensures all humidity values are extremely high (at least 95%) to simulate very moist conditions; if testing **"high wind"**, it will multiply the wind speeds by 5 to create an extreme wind scenario-. These hypothesis switches allow studying how drastically different environmental factors influence the fire spread, by essentially "locking in" an extreme condition.
At the end of initialization, we have a wildfire simulation ready to run: a grid with some cells burning, environment maps for fuel and weather conditions, and parameters set for the simulation loop.

Workflow Step 4: Wildfire Spread Simulation Loop

Once initialized, the core simulation of wildfire spread occurs in a time-stepped loop (within WildfireSimulation.simulate_spread). The model advances through discrete time steps (up to the max of 24, or until the fire dies out). At each step, the following happens:

1. **Evolve Environmental Conditions:** The simulation updates the weather factors to reflect changes over time. This is done with an **auto-regressive (AR) modeling approach** for each factor (temperature, humidity, wind) to introduce realistic temporal persistence. Essentially, the new value = (previous value * 0.9) + (random variation * 0.3). For example, the temperature grid at the next step is generated by taking 90% of the current temperature and adding a random perturbation (drawn from a normal distribution around 25°C with some std) weighted at 30%. This keeps temperatures fluctuating but not jumping wildly – yesterday's weather strongly influences today's. The humidity grid is updated similarly: 90% of the old humidity plus a random draw around 50% humidity-, then clipped to [0,100]%. Wind direction changes are made more gradual (small random perturbation around 0°) and wrapped around 360°, while wind speed is updated with a random component around a baseline (e.g. 10 m/s) and kept non-negative. This AR model is a **predictive component** – it reflects that environmental conditions have memory and continuity, which is scientifically accurate for weather. It also means the simulation can capture

phenomena like a prolonged dry spell or a windy period that lasts several steps. (If a hypothesis like high humidity is active, the code may enforce constraints after updating – e.g., maintaining humidity very high as noted earlier.)

2. **Compute Spread Probabilities:** Given the current state (which cells are burning) and all the environmental factors (both static and updated), the model computes the probability of fire spreading to each neighboring cell. This is where the scientific wildfire spread logic comes in, inspired by the Rothermel fire spread model. For each cell that is currently burning, the code examines its neighbors in the 8 directions (including diagonals). For each neighbor that is not already burned, it calculates a **spread probability** based on several contributors:
    - **Base Ignition Likelihood:** a product of the cell's vegetation flammability and the historical ignition propensity at that location. If a cell has highly flammable vegetation (e.g. dense grass) and was often a fire hotspot historically, its base probability will be higher. If either factor is zero (e.g. water body or never recorded a fire), base probability is zero.
    - **Temperature:** higher temperatures dry out fuels and promote fire, so a higher normalized temperature in that cell increases the spread chance. In code, this is added proportionally (weighted contribution).
    - **Humidity:** high humidity suppresses fire (moisture prevents ignition), so the model uses (1 - normalized humidity) as a factor – meaning lower humidity (closer to 0, so 1 minus that is large) yields a bigger boost to fire spread probability. Effectively, as humidity rises, this term diminishes. Additionally, a further dampening multiplier is applied based on absolute humidity later (to ensure very high humidity almost never allows fire).
    - **Wind Speed:** stronger winds aid fire spread by carrying embers and providing oxygen, so the higher the wind speed at that cell, the higher the probability contribution.
    - **Elevation/Slope:** steeper uphill terrain slows fire spread (fire struggles to move downward or against slope), whereas downhill or flat terrain offers less resistance. The model encodes this by using (1 - normalized elevation) as a factor– implying that higher elevation areas (which often correspond to uphill from the fire) get a lower probability. This is a simplification treating higher altitude as a proxy for being "uphill" or rugged terrain.
    - **Wind Direction Alignment:** The only non-linear factor, wind direction, is handled after the above linear combination. If the direction of the wind at the neighbor cell aligns with the direction from the burning cell to this neighbor, it greatly increases spread likelihood. The code calculates the angle between the wind vector and the vector pointing from the fire to the neighbor, and uses the cosine of that difference. If the wind is blowing directly toward the neighbor (angle ~0°, cos≈1), the probability is effectively multiplied by ~1 (no reduction); if the wind is opposite to the spread direction (180° off), the cosine becomes -1 and the code floors this influence at 0 (no spread help). Thus, wind can either favor a spreading direction or, if blowing against, make it much less likely for fire to move that way.

    - **Minimum Probability Floor:** After combining all these influences, the model imposes a minimum spread probability of 0.01 (1%) for any neighboring cell that is otherwise reachable. This ensures there's always a slight chance of spread even in poor conditions, preventing the fire from getting "stuck" completely due to a deterministic zero probability.

All these computations produce a matrix of spread probabilities for the next set of fires.

3. **Stochastic Spread to Neighbors:** Using the probability map, the simulation then decides which cells actually catch fire this step. This is done by comparing each probability to a uniform random number draw (0–1) for that cell. If the random draw is less than the spread probability, that neighbor will ignite (i.e. become a new fire cell). This Monte Carlo approach introduces randomness in the spread outcome: even a cell with, say, 80% probability might by chance avoid ignition in one simulation, or a cell with 5% probability might get ignited in another run. This reflects the unpredictable nature of ember travel, fuel connectivity, etc., in real fires. All new ignitions are marked as burning (state = 1) in the fire grid.
4. **Update Fire State:** Cells that were burning in the previous step are now marked as **burned** (state = 2) to indicate they have finished burning and will remain burned out (the fire has passed). The newly ignited neighbors become the burning cells (state = 1) for the next iteration. The simulation records the total count of burned cells after this update and appends it to the burned_areas history list. This list thus tracks how the burned area accumulates over time.
5. **Convergence Check (Fire Extinction Criteria):** If the simulation is not in a special hypothesis mode (i.e. we are doing a normal run), it checks whether the fire has effectively stopped spreading. It calculates the relative change in burned area over the last few time steps (e.g. last 5 steps) and if all those changes are below a small threshold (meaning the fire isn't growing appreciably anymore), it breaks out of the loop early. This serves as an automatic stopping condition to simulate the fire dying out before the maximum time steps, saving computation if, for example, the fire burns out all available fuel or is suppressed by conditions.

This loop repeats until either the fire is no longer spreading or the maximum number of time steps is reached. The outcome of a single simulation run is typically the final fire grid (showing which cells burned) and the time series of burned cell counts.
Scientific methodology note: The spread model used combines deterministic factors (like fuel and weather influence based on fire science models such as Rothermel's equation for spread rate) with randomness to capture uncertainty. The **auto-regressive weather modeling** is essentially a first-order AR(1) process ensuring temporal correlation in temperature/humidity – a realistic touch since real weather doesn't reset every hour. The probabilistic spread ensures that even under identical conditions, two simulations might yield different burn patterns, which is why multiple runs are needed for robust insights.
Workflow Step 5: Monte Carlo Simulations and Convergence Analysis

Because wildfires are chaotic and the model includes many random components (initial seasonal conditions, humidity/wind sampling, stochastic spread decisions), a single simulation run can only tell one possible story. The project therefore employs a **Monte Carlo approach**: running many simulations and analyzing the distribution of outcomes. The run_simulation function in wildfire_simulation.py automates this. Key aspects of this phase:

- **Multiple Runs in Parallel:** The code can run a large number of simulations (e.g. 1500 runs) in parallel using Python's multiprocessing Pool. Each run calls the single-run simulation described above, potentially with a different random seed leading to different outcomes. By leveraging multiple CPU cores, it significantly speeds up the computation of many trials.
- **Aggregation of Results:** After all runs, the results (such as total burned area and number of time steps for each run) are collected. The code then calculates summary statistics: mean and standard deviation of burned area (in cells), mean and std of the percentage of the grid burned. This provides an estimate of the expected fire coverage and variability under the given conditions. For example, it might report that on average 60% of the grid burns with a standard deviation of 5%, indicating a fairly consistent outcome.
- **Standard Error and Confidence Interval:** With many runs, the standard error of the mean burned area percentage can be computed (std dev divided by sqrt of runs). The script prints a 95% confidence interval for the burned area percentage to indicate the range in which the true mean is likely to lie. This is a statistical measure giving credibility to the results.
- **Monte Carlo Convergence Check:** A critical question is how many simulation runs are enough to get a stable estimate. The code includes a convergence check that monitors the moving average of the burned area percentage as more runs are added. It looks at the relative change in this moving average over the last N runs (e.g. last 10) and also checks if the standard error is below a threshold (e.g. 0.1%). If both the moving average has stabilized (changes below 5% per run, for instance) and the standard error is sufficiently low, the simulation declares that the Monte Carlo has **converged**, meaning additional runs would likely not change the average significantly. The code then can report convergence achieved with the final moving average value. If not converged, it would report that more runs might be needed (or simply that convergence was not reached within the runs performed).
- **Visualization:** The script even provides a function to plot the convergence behavior – plotting each run's burned percentage and the moving average over runs, as well as the relative change per run. This helps in verifying that the Monte Carlo simulation results have stabilized. It's a scientific way to ensure that the number of trials was sufficient for reliable statistics.

Through this Monte Carlo process, the project turns many random simulations into meaningful aggregate insights. We get not just one possible outcome, but an understanding of the distribution of possible wildfire extents under the modeled conditions, increasing confidence in any conclusions drawn.

Key Design Elements and Motivations

Random Seasonal Quadrant Selection (Variability of Conditions)

One deliberate design choice is the random selection of the seasonal **quarter (Q1–Q4)** for each simulation run. This introduces variability in initial environmental conditions – essentially sampling different **climate regimes** (e.g. winter vs summer conditions). The motivation is to avoid biasing the simulation to a single set of conditions and to test the fire behavior across a spectrum of plausible scenarios. In reality, wildfires can occur in different seasons with different weather; by randomizing the quarter, the simulation's outcome statistics incorporate that seasonal uncertainty. Over many Monte Carlo runs, the effect is that some fraction of runs were "dry season" fires, others "wet season," etc., painting a more comprehensive picture of fire risk.

Stochastic Humidity and Environmental Factors

Instead of using a fixed humidity or wind scenario, the simulation draws these values from historical distributions for the given quarter. This random humidity value selection means each run might start with, say, 40% average humidity, another with 70%, etc., reflecting day-to-day natural variation. The **scientific reason** is that ignition and spread are highly sensitive to these factors; using a distribution-based random draw ensures that the simulation explores the range of difficulty/ease of burning. Low humidity runs will show aggressive fire growth, whereas high humidity runs might show fires fizzling out – both are important outcomes to capture. Similarly, randomizing wind speeds and directions per run ensures some fires happen on calm days and others under windy conditions. These random initial draws are anchored in real data (fitted to climate records) to keep them realistic, but add the necessary spread in possible conditions.

Within the simulation steps, randomness continues to play a role: the chance-based spread to each neighbor means even under the same overall conditions no two fires burn exactly the same way. This stochastic treatment mirrors real wildfire behavior where many small factors (like spark travel distance, local fuel moisture variations) introduce randomness in which patch ignites next. The **motivation** for all these random elements is to capture the **uncertainty and variability** inherent in wildfire events, rather than yielding one deterministic result.

Auto-Regressive Weather Modeling (Predictive Continuity)

The use of an **auto-regressive (AR) model** for updating temperature, humidity, and wind each time step is a key predictive component in the simulation's design. Rather than sampling new independent weather conditions at each step, the AR approach uses the previous state to inform the next. This models the persistence of weather patterns – e.g. if a fire starts on a hot dry day, it's likely to stay relatively hot and dry for the next few hours. By tuning the persistence factor (set to 0.9 in the code) and variability (0.3), the simulation achieves a balance where conditions evolve gradually and realistically. The **motivation** here is scientific realism: fires often last several hours or days, during which weather can change, but usually smoothly rather than erratically. The AR model effectively acts like a simple weather forecast within the simulation, predicting that the next timestep will be similar to

the last with slight randomness. This improves the realism of fire spread – for instance, a wind that suddenly reverses direction in reality is rare, and thanks to the AR design, the simulated wind also won't do something so drastic without cause.
Moreover, this predictive modeling allows exploration of scenarios like prolonged drought or a sudden weather change by adjusting the persistence or variability. It was noted in the code comments that these formulas were developed due to limited documentation on incorporating weather in wildfire models, indicating a pragmatic solution to inject believable weather dynamics.

Probabilistic Spread Model (Fuel and Weather Integration)
The fire spread calculation combines multiple factors linearly with weights, following principles of wildfire science (fuel, moisture, wind, terrain). This design was inspired by the Rothermel spread model, which traditionally takes inputs like fuel type, fuel moisture (influenced by humidity and recent rain), wind speed, and slope to determine fire spread rate. In our simulation, these correspond to vegetation (fuel), humidity (moisture), wind (speed and direction), and elevation (slope proxy) influences. By assigning weights to each, the model allows tuning of their relative importance. For example, the code gave a high weight to humidity in one configuration, meaning moisture content can strongly inhibit fire spread (which aligns with fire science). The inclusion of a wind direction factor multiplies the probability by a factor between 0 and 1 depending on alignment, which is a way to encode wind-driven fire behavior (fires spread faster downwind). The **motivation** for this approach is to strike a balance between physical realism and computational simplicity: a fully physical model would be very complex, so a weighted probabilistic model is used to approximate those effects in a simpler way. Randomness is then applied on top of these probabilities to simulate the chance nature of fire catching or not catching despite conditions.

Monte Carlo Simulation for Robust Results
Finally, the choice to run many simulations and analyze their outcomes is a methodological design decision. Wildfire outcomes are inherently probabilistic in this model; any single run could be an outlier (either almost no spread or near-total burn). By running hundreds or thousands of simulations, the project treats the problem statistically, asking **"on average, how much burns and with what variability?"**. The convergence checking using moving averages and standard error is motivated by ensuring scientific rigor – it's similar to ensuring one has a large enough sample size in an experiment. When the average burned area stabilizes within a small margin of error, one can be confident in the results. This approach reflects good scientific practice: rather than trusting one simulation, it uses the power of repeated random sampling (Monte Carlo) to draw conclusions with confidence intervals.
In summary, each random or predictive element in the design has a clear purpose:

- Random selection of initial conditions (season, humidity, wind) broadens scenario coverage.
- Autoregressive updates of conditions maintain realism over time.
- Probabilistic spread with weighted environmental factors injects real-world uncertainty and multi-factor influence.

- Repeating simulations many times and checking convergence ensures the findings (like average burn area) are reliable and not an artifact of random chance.

Integration of Modules in the Overall Workflow

All the above components come together in a cohesive workflow:

1. **Data Ingestion:** The EnvironmentalFactors module reads external data (climate records, vegetation maps, elevation API) and produces the necessary input grids. The fire analysis script reads historical fire data and outputs ignition hotspot grid.
2. **Simulation Initialization:** The WildfireSimulation uses outputs from those modules (by instantiating EnvironmentalFactors and loading the ignition grid) to set up each simulation run with consistent data inputs.
3. **Simulation Execution:** The core loop uses the data each step (reading the updated weather grids, referencing static grids for vegetation and elevation) to calculate spread. Here the integration is visible: e.g., it normalizes and plugs in the temperature, humidity, wind, elevation values from the EnvironmentalFactors outputs into the probability model. Without the EnvironmentalFactors providing realistic grids, the simulation would have to rely on synthetic or constant values. Likewise, the ignition_points from the fire analysis directly influence how the fire starts and thus the outcomes.
4. **Post-Simulation Analysis:** The results of many simulation runs could be further analyzed or visualized. (For instance, one might compare runs under different hypothesis settings: high humidity vs normal to quantify the difference, which the framework easily supports by toggling the hypothesis parameter and re-running.)

Throughout, there is a clear separation of concerns: one part generates the environment, another part runs the fire logic, and another collates statistics. This modular design makes it easy to adjust one piece (say, use a finer grid or a different climate dataset) without overhauling the entire system – a good software engineering practice. From a scientific perspective, it ensures that each aspect (fuel, climate, fire dynamics) is modeled with the appropriate data or method and then all pieces are synthesized in the simulation.

Conclusion

In this wildfire simulation project, the workflow starts from data-driven environmental setup, moves through a carefully constructed simulation of fire spread that merges scientific fire behavior principles with randomness, and ends with a Monte Carlo analysis to derive meaningful insights. The design choices – using auto-regressive weather updates, random draws from climate distributions, and random quadrant (season) selection – are motivated by the desire to simulate reality as closely as possible. Weather and ignition sources in nature are never fixed or fully predictable, and the simulation embraces that variability. By integrating multiple scripts and data sources, the project ensures that the simulation is not operating in a vacuum but is grounded in real environmental context (e.g., real climate patterns and historical fire occurrence).

Overall, the **software implementation** (parallel processing, use of NumPy for calculations, caching of generated grids, etc.) enables efficient execution of what is fundamentally a **scientific methodology**: experiment with many simulated fires under varied conditions, observe the outcomes, and draw conclusions with statistical confidence. The result is a robust workflow that can be used to test wildfire behavior hypotheses or to estimate fire spread extents under different environmental scenarios in a given region, with each module contributing vital pieces to the puzzle.