Design Decisions:

- Decided to club spatial and temporal check because they are related and checking them separately using different methods would result in less efficient and modular code.
- Code consists of 3 main parts:
  - Flight Simulator (All functions necessary for waypoints and timestamps -> Mission Params and Drone State)
  - Conflict Detector (Defines conflict detection logic)
  - Example.py (simulates a real scenario)

Code Structure:

- requirements.txt - generated by AI that contains all the specific version requirements for the libraries used in this project.
- models.py defines all the important data structures specific to this project. Data classes and Enum was used to implement it cleanly and efficiently. (implementation here was assisted by AI while implementing other parts of code)
- flight_path_simulator.py contains all helper functions for simulation of the mission and conflict detection. This part was also built in assistance with AI.
- conflict_detector.py contains conflict detection logic using safety buffers, and creating a temporo-spatial check (instead of a spatio-temporal check to ensure realistic representation).
- visualisation_4d.py contains visualisation code using matplotlib to generate both 3D and 3D+time(4D) visualisations.
- Waypoints.json contains the hardcoded drone waypoint, timestamp and speed data for other drones. This was created by trial and error to ensure all of them are conflict-free between themselves.
- example.py contains the main execution code to simulate the actual scenario using the above helper modules. It creates the missions from the drone 1 waypoints and waypoints.json, feeds the missions into the conflict_detector.py functions, and then uses both mission data and conflict data to generate visualisations.

Scalability Discussion:

Currently, the system is a single-threaded Python prototype with everything happening in-memory and displayed with Matplotlib. It's perfect for prototyping, but in order to accommodate thousands of drones in actual airspace, it must scale in both performance and architecture.

The first step is breaking things into modular microservices so each core function (like mission ingestion, conflict detection, or visualization) can run independently and scale as needed. We'd containerize the services and deploy them using something like Kubernetes, allowing us to auto-scale based on load. Communication between services would be handled through message queues (probably Kafka) so everything can stay decoupled and responsive.

To handle in real-time, we'd stream in incoming missions using Kafka and something like Spark or Flink to compute that data in real-time. We'd use sliding windows as well to verify conflicts only within pertinent time windows, not across the whole dataset. Redis would be a cache layer to accelerate repeated access to frequently accessed data such as recent mission trajectories or recent conflict areas.

On the storage front, mission and waypoint information could reside in distributed databases such as MongoDB or Cassandra with geospatial indexing to accommodate quick queries. A time-series database could be used to store historical conflict data and trend patterns in time. The entire thing would get shared by region and time so it stays query-efficient at scale.

The underlying conflict detection logic would also be optimized via spatial indexing (such as R-trees) and parallel processing. In the long term, we can even incorporate ML-based conflict prediction to pre-emptively mark dangerous paths. Monitoring would be done with Prometheus + Grafana, and the system would be made fault-tolerant through retries, circuit breakers, and fallback handling.

The goal is to eventually support over 10,000 concurrent drone missions, process upwards of 100,000 waypoints per second, and detect conflicts in under 100ms—while keeping latency low and uptime high. We'd scale in phases: start by containerizing and adding basic monitoring, then implement real-time data handling and caching, shift to distributed databases, and finally bring in AI and advanced automation.