4. Evaluate the Limitations of the Testing Process, Using Statistical Methods Where Appropriate, and Summarise Outcomes

4.1 Identifying Gaps and Omissions in the Testing Process

A key gap lies in performance testing for the first request. Although subsequent requests exhibit lower latency, there is no dedicated mechanism to statistically quantify or predict worst-case scenarios. In the code, particularly in the fetchRestaurants() and fetchNoFlyZones() methods of the OrderValidation class, there is no caching to mitigate initial call delays, leaving the first request vulnerable to high response times. Additionally, security testing is minimal; for instance, the calcDeliveryPath method in DeliveryController does not perform strict input sanitization, making it susceptible to malformed inputs that could degrade performance.

4.2 Identifying Target Coverage/Performance Levels

The initial goal was to keep average latencies below **200ms** for all requests, including the first. Our existing Inteliji coverage tools identify a target of **90%** branch coverage for the core pathfinding logic (calculatePath()), but only **80–85%** is currently achieved. The code in NoFlyZone.intersects() and CentralArea.contains() has partial tests, particularly for boundary edges where floating-point precision might lead to missed intersections.

4.3 Comparing the Carried-Out Testing with the Target Levels

We achieved near-full coverage in OrderValidation but only partial coverage for path generation under extreme or rarely encountered conditions, such as multiple overlapping no-fly zones. We also lack detailed performance metrics: while we have manual observations of 400ms for the first request, we have not introduced statistical methods like collecting standardized latencies (mean, variance) across multiple runs. Without these measurements, we cannot definitively confirm whether the system meets the 200ms target in all environments.

4.4 Discussion of What Would be Necessary to Achieve the Target Levels

To close these gaps, the testing process would require:

1. **Caching/Preloading** in fetchRestaurants() and fetchNoFlyZones() to reduce the first-request overhead and gather latency statistics (e.g., average, max) over multiple runs.
2. **Enhanced Security Testing**, such as fuzzing the /calcDeliveryPath endpoint with invalid inputs to detect potential performance bottlenecks or vulnerabilities.
3. **Statistical Profiling** of performance to collect formal metrics (confidence intervals, standard deviation) around response times. Tools like JMeter or Gatling can automate repeated runs of the same test scenario and compute these statistics.
4. **Increased Coverage** in NoFlyZone and CentralArea by introducing more boundary-condition tests, especially around floating-point precision or overlapping polygons.

Overall, addressing these limitations would bring the system closer to its target performance thresholds, ensure broader coverage of the path-calculation routines, and solidify security under more varied inputs.