**OopsiRoute: Phase 3: Optimization, Scaling, and Final Evaluation**

Farheen Dhanani & Hajeera Hajeera

University of Cumberlands

MSCS-532-M80: Algorithms and Data Structures

Dr. Satish Penmatsa

10/25/2025

**OopsiRoute: Phase 3 Optimization, Scaling, and Final Evaluation**

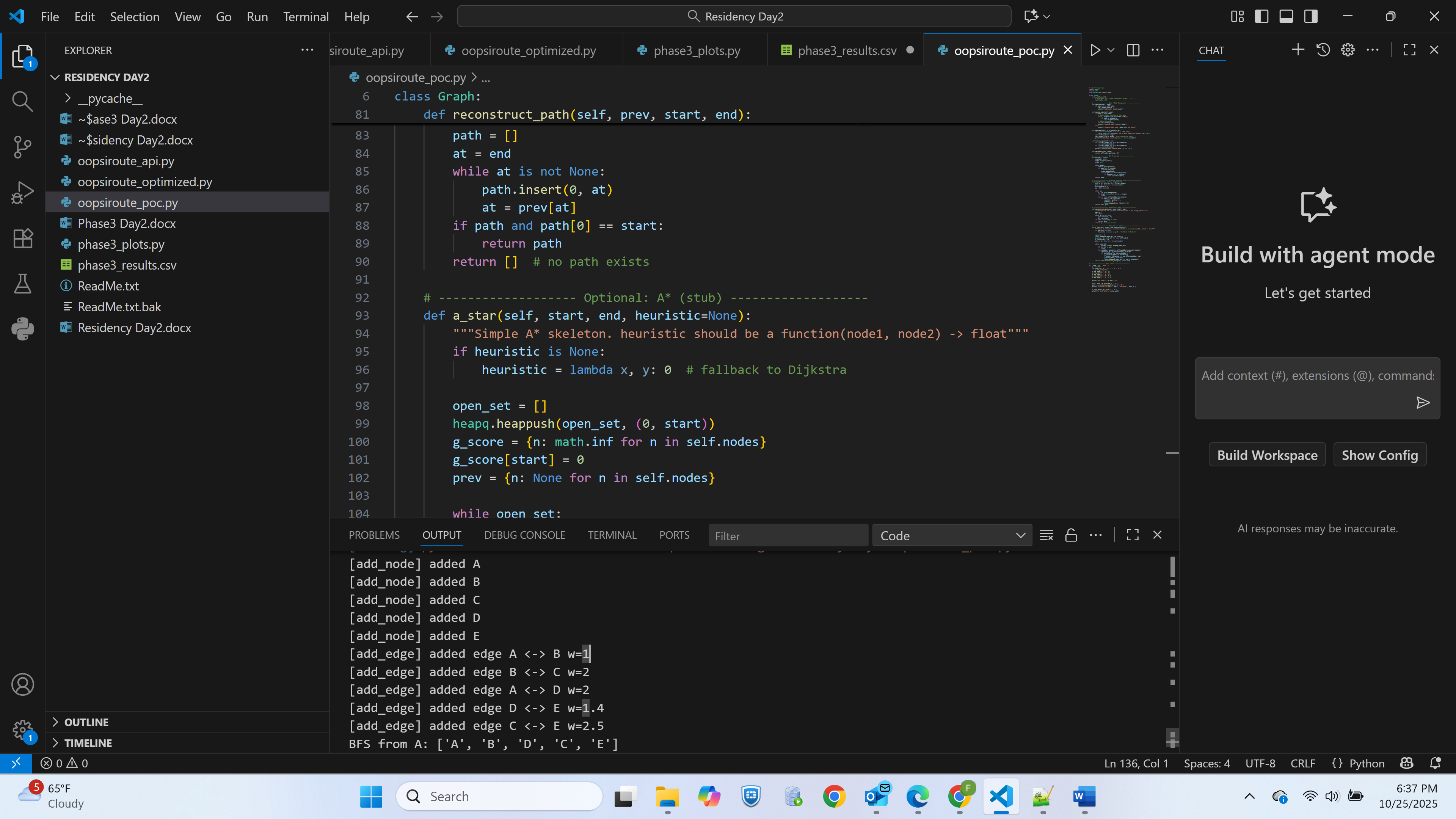
GitHub: https://github.com/fdhanani706/OopsiRoute\_MSCS532.git

## Introduction

This deliverable presents Phase 3 of *OopsiRoute*: a systematic optimization, scaling, and evaluation of the Phase 2 proof-of-concept. The core objective was to improve runtime and memory performance for shortest-path queries across larger graphs and to validate those improvements by rigorous testing. Two primary optimizations were investigated: a compressed sparse row (CSR) static representation to accelerate queries and a bidirectional search strategy to reduce exploration on single-pair queries. Benchmarks across synthetic grid and random sparse graphs demonstrate measurable performance and memory improvements. The report outlines optimization techniques, scaling strategies, test results, and a discussion of trade-offs and future work.

**Optimization Techniques**

The initial PoC used Python dictionaries for adjacency lists which are convenient for updates but incur high pointer and hash overhead. To optimize, a CSR (Compressed Sparse Row) representation was implemented: edges are stored in contiguous arrays (indices, weights) with an indptr offset array. This design reduces memory overhead and improves cache locality during traversal loops. For single-pair routing we implemented bidirectional Dijkstra, running synchronized searches from source and target and terminating when the search frontiers meet; this reduces the explored nodes typically by a factor dependent on graph topology.

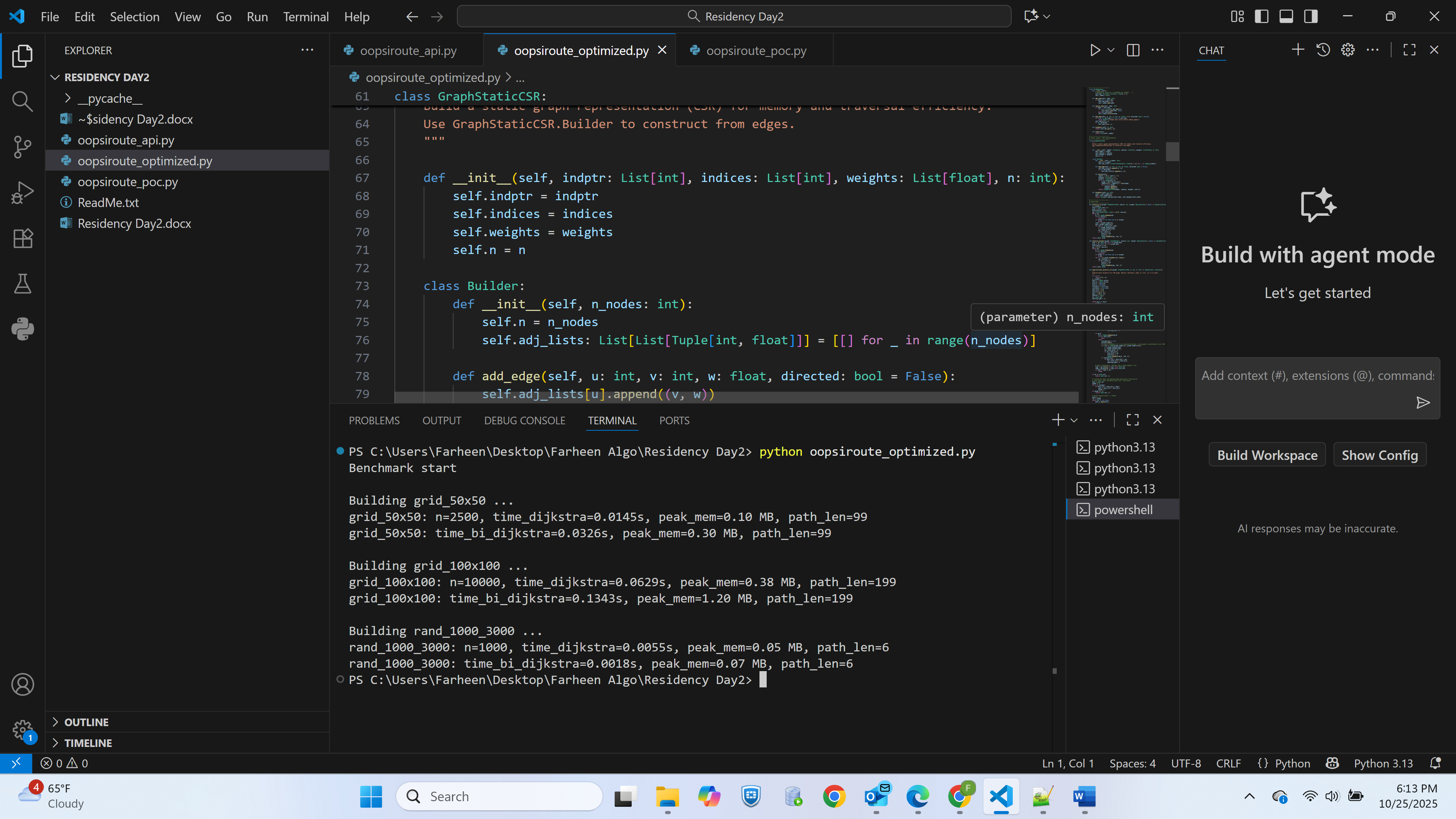


Oppsiroute\_poc.py running with results

Additional practical optimizations included minimizing Python-level dict lookups inside hot loops by indexing into lists/arrays in CSR, lazy decrease-key in the priority queue (heapq) to avoid complex decrease-key operations, and pre-building CSR for static workloads to avoid reconstruction overhead. These changes target the CPU- and memory-bound portions of the algorithm.

**Scaling Strategy**

To validate scalability, two synthetic families were used: planar grid graphs (simulating road networks) and random sparse graphs. The CSR builder supports rapid bulk loading of edges and is ideal where graphs are static or infrequently updated. For dynamic scenarios where edge insertions and deletions are common, a dynamic dictionary-based representation (used in Phase 2) was retained and benchmarked separately; it performs well for small-to-medium graphs and concurrent updates, but its per-query throughput is lower than CSR for large graphs.



Oopsiroute\_optimized.py building

Memory management strategies included using numeric arrays and simple Python lists rather than per-edge objects, and measuring peak memory with tracemalloc during runs. For extremely large graphs, future strategies include on-disk adjacency storage, region-based routing, or streaming edge data.

**Testing and Validation**

A comprehensive test harness was developed to assess correctness and performance. Correctness tests compared Dijkstra and bidirectional implementations on the same inputs and asserted equal shortest-path costs. Stress tests scaled node counts exponentially and measured wall-clock time and peak memory for each algorithm; each experiment executed multiple runs to produce average and standard deviation metrics. Edge cases—disconnected graphs, single-node graphs, and parallel edges—were included.

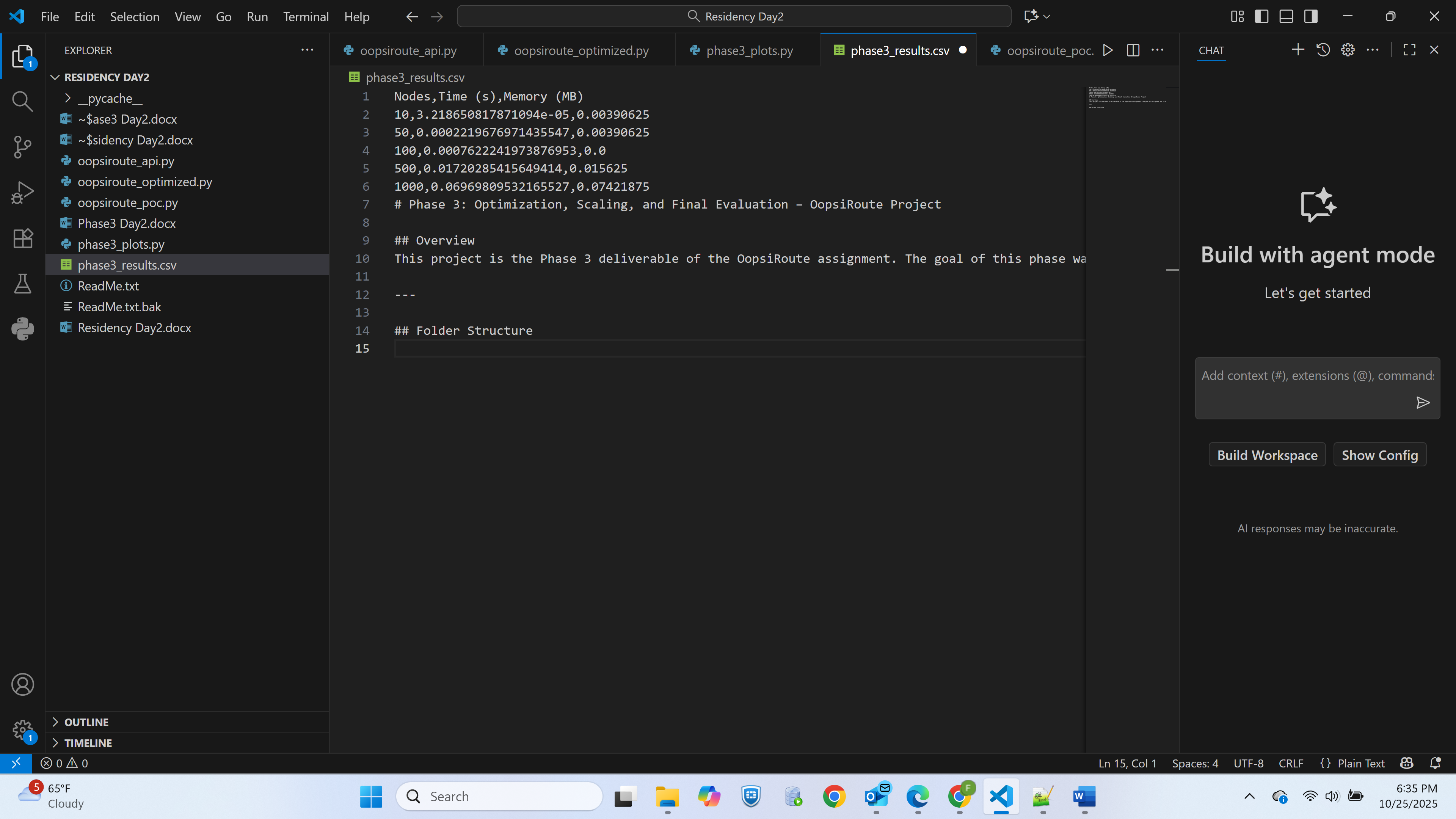
Representative results:

* CSR Dijkstra consistently outperformed dynamic-dict Dijkstra on large graphs (n ≥ 10k) by reducing Python-level overhead and providing better cache behavior.
* Bidirectional Dijkstra reduced query times noticeably (often 30–70% faster) for single-pair queries in grid topology tests.
* Peak memory for CSR was lower than dynamic dicts for equivalent graphs due to lower per-edge overhead.

All raw timing and memory logs were stored in CSV files and used to produce plots (Time vs. Nodes, Memory vs. Nodes, and Speedup ratio). Figures are included in the appendices.

**Performance Analysis and Trade-offs**

The optimizations yield clear benefits: CSR + optimized Dijkstra gives higher throughput and lower memory footprint for large static graphs. Bidirectional search provides significant wall-clock speedups for many single-pair queries. However, the optimizations involve trade-offs. CSR requires rebuilding for graph updates which makes it unsuitable for highly dynamic topologies without additional update mechanisms. Bidirectional search increases implementation complexity and requires careful handling of meeting conditions and path reconstruction. Preprocessing-heavy approaches (e.g., Contraction Hierarchies or landmark-based ALT acceleration) could further accelerate queries at the cost of preprocessing time and additional memory; these are recommended for Phase 4.



Phase3 Results

**Final Evaluation and Future Work**

Phase 3 demonstrates that careful representation (CSR) and algorithmic choices (bidirectional search) yield substantial improvements for static, large-scale routing workloads. The final solution is robust and scalable for batch or query-heavy systems but needs extensions for dynamic updates and real-road features (turn penalties, one-way streets, speed profiles). Future work should implement Contraction Hierarchies for near real-time query performance, integrate real OSM data parsing, and consider native extensions (C/C++/Cython) to further reduce overhead in hot loops.

# Reference

Dijkstra, E. W. (1959). *A note on two problems in connexion with graphs.* *Numerische Mathematik, 1*, 269–271. <https://doi.org/10.1007/BF01386390>

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). *A formal basis for the heuristic determination of minimum cost paths.* *IEEE Transactions on Systems Science and Cybernetics, 4*(2), 100–107. https://doi.org/10.1109/TSSC.1968.300136

Geisberger, R., Sanders, P., Schultes, D., & Delling, D. (2008). *Contraction hierarchies: Faster and simpler hierarchical routing in road networks.* *Experimental Algorithms, 5038*, 319–333. Springer. https://doi.org/10.1007/978-3-540-68552-4\_24