جامعة الأخوين

ⵜⴰⵙⴷⴰⵡⵉⵜ ⵏ ⵓⵅⴰⵡⵉⵏ

AL AKHAWAYN UNIVERSITY

School of Science and Engineering

Milestone 1: Project Inception

Realized by:

Ayoub AKARKOUZ < 78987 >

Under the supervision of:

Pr. Asmaa MOURHIR

~ Spring 2026 ~

Table of Contents

AI-Assisted Symbolic Optimization for Strategic Facility Network Design

1. Framing the Business Idea as an ML Problem

1.1 Business Case Description

1.1.1 Context

Facility location decisions are central to strategic planning: organizations choose which facilities to open and how to assign customers to minimize fixed and servicing costs while meeting service requirements. This occurs in fulfillment networks, telecom infrastructure, EV charging deployment, cloud data centers, and healthcare service planning. The decision is formalized by the Uncapacitated Facility Location Problem (UFLP):

$$min \sum_{i \in F} f_i y_i + \sum_{i \in F} \sum_{j \in C} c_{ij} x_{ij}$$

Subject to each customer being assigned to exactly one facility and assignments only to open facilities, with $y_i, x_{ij} \epsilon \{0,1\}$.

UFLP is NP-hard and widely used in combinatorial optimization benchmarking [1]. In practice, the strategic bottleneck is often not solver performance but the time and expertise required to formalize the optimization model itself. In this project, business impact will be measured through quantifiable proxies such as:

- Modeling time reduction (minutes from instance load to first feasible model vs manual formulation)
- Manual intervention reduction (number of prompt edits or code patches required to reach feasibility)
- Iteration speed (time to modify and re-solve under new cost scenarios).

1.1.2 Core Business Pain Point

Although solvers such as CPLEX, Gurobi, and OR-Tools can solve UFLP effectively once formulated, model formulation is the main bottleneck in enterprise settings. Turning business requirements into a correct MILP requires specialized OR expertise, increases cost, slows scenario iteration, and creates risk of constraint or data-mapping errors. The problem is therefore not primarily solver speed, but modeling time and correctness.

1.1.3 Proposed Solution

We propose an AI-assisted modeling pipeline where a Large Language Model (LLM) generates solver-compatible MILP code from structured instance input. The generated model is executed by a classical solver, and correctness is assessed through feasibility checks and objective verification. This follows the LLMFP paradigm, where LLMs produce formal programs that are executed and verified by external solvers [2], and can optionally be refined using solver feedback [6]. Unlike LLM-as-optimizer approaches such as LEO [7], the LLM is restricted to model generation, while solution quality and feasibility are guaranteed by solver-based verification.

1.2 Business Value of Using ML

The value proposition is modeling automation, not solver acceleration. Automating symbolic optimization generation can:

- Reduce time from data to first feasible model
- Lower dependency on OR specialists
- Reduce modeling errors via structured constraint generation
- Improve scenario iteration speed through rapid regeneration of models

This supports scalable, reusable optimization modeling pipelines for non-expert users.

1.3 Data Overview

1.3.1 Dataset Source and Provenance

We use the OR-Library benchmark suite curated by Beasley [1], specifically the "Uncapacitated warehouse location" collection. OR-Library is a widely used repository of standardized OR test instances, supporting reproducible algorithmic evaluation.

1.3.2 Benchmark Scope and Scale

The uncapacitated warehouse location collection contains 15 instance files (cap71 - cap74, cap101 - cap104, cap131 - cap134, capa, capb and capc). Each instance provides Facility count m, Customer count n, Fixed costs f_i and Allocation costs c_{ij}. Moreover, instances span increasing levels of complexity. For example:

- cap71: $m = 16$ facilities, $n = 50$ customers
- Larger "cap*" instances contain significantly more decision variables

From a practical standpoint, the full collection is lightweight (≈ 3.7MB total), making it convenient for reproducible experiments and CI-style benchmarking. Crucially, OR-Library provides known optimal objective values (uncapopt.txt), enabling exact optimality gap computation.

These instances remain active in recent research, including ABPEA [3] and a hybrid GA + simulated annealing approach [4], which confirms the benchmark's research relevance.

1.3.3 Input Structure and Semantics

Each instance is provided as a plain text file with a fixed, machine-parsable structure:

- First line: m (number of candidate facility/warehouse locations) and n (number of customers).
- For each facility $j \epsilon\{1,\ldots,m\}$: a pair (capacity, fixed opening cost).
- For each customer $j \epsilon\{1,\ldots,n\}$: a demand value followed by m allocation costs (shipping/service cost of assigning customer j to facility i).

Although the files include capacities and demands due to their origin in capacitated variants, OR-Library explicitly states these quantities should be ignored for the uncapacitated version (UFLP). Thus, the effective optimization input is a fixed-cost vector f_i and an allocation-cost matrix c_{ij}

1.3.4 Output, Ground Truth, and Verifiability

A solution consists of binary facility-opening decisions $y_i \epsilon\{0,1\}$, customer-to-facility assignments $x_{ij} \epsilon\{0,1\}$, and a total objective value.

Ground truth optimal objectives from uncapopt.txt allow strict verification via feasibility checking and exact optimality gaps [1], which is essential for evaluating LLM-generated symbolic models. This enables an exact optimality gap computation, feasibility verification, and reproducible research-grade benchmarking.

1.4 Project Archetype

This system can be formally categorized as an "LLM-as-modeler" architecture, where the LLM generates symbolic optimization programs while a classical solver performs execution and verification. It belongs to the following AI system archetypes:

- Neuro-Symbolic Optimization System (generation → formal solve)

- LLM-Augmented Decision Systems (LLM performs model construction, not direct decision-making)
- Verification-Guided AI (Solver verifies feasibility and optimality)
- Code Generation for Optimization (Inspired by LLMFP methodology)

It differs from LLM as optimizer frameworks such as LEO [7] by enforcing solver-based correctness.

1.5 System Architecture Overview

Unlike learning-to-optimize approaches that directly construct solutions, our system separates symbolic generation from optimization execution. The workflow is as follows:

- Structured instance data are parsed from OR-Library.
- A constrained prompt template encodes the problem specification.
- The LLM generates solver-compatible OR-Tools code (symbolic MILP model).
- The generated code is executed in an automated environment.
- A classical solver computes the solution.
- Results are evaluated against the known optimal objective values (uncapopt.txt).

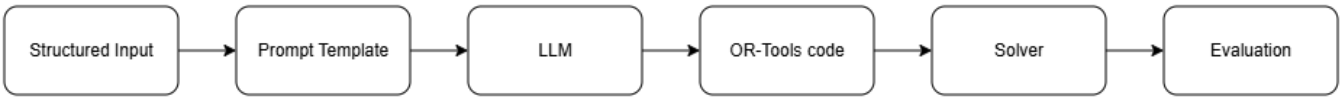

Figure 1: End-to-end architecture of the proposed LLM-assisted symbolic optimization pipeline for UFLP

2. Feasibility Analysis

2.1 Literature Review

2.1.1 Classical Facility Location Optimization

UFLP is a canonical benchmark in OR with extensive research across exact MILP methods, approximation algorithms with guarantees [9], relaxations, and hybrid metaheuristics. OR-Library instances remain widely used due to standardized structure and known optimum values [1]. Recent work includes ABPEA [3] and a hybrid GA + simulated annealing method [4], demonstrating strong performance on cap instances. However, classical research assumes the model is manually written; the modeling process is typically treated as external.

2.1.2 Machine Learning for Combinatorial Optimization

Recent years have witnessed increased interest in applying machine learning to combinatorial optimization problems. Approaches include learning heuristics or branching strategies, neural combinatorial optimization, policy-gradient methods for solution construction, learning-based parameter tuning…

These approaches attempt to approximate or accelerate optimization, often replacing or augmenting classical heuristics. However, a key limitation of this research direction is that it primarily targets solution quality or speed, not the formal construction of optimization models. The underlying assumption remains that a correct mathematical formulation is already available. In contrast, the bottleneck in many enterprise applications lies not in solver speed but in model formalization, validation, and iterative specification. This gap motivates investigation into AI systems capable of generating formal optimization programs automatically.

2.1.3 LLM-Based Symbolic Planning and Optimization

LLMs have enabled structured code generation and symbolic reasoning. LLMFP proposes generating a formal program, executing it with a solver, verifying correctness, and iteratively refining based on feedback [2], with related results on rigorous real-world planning via formal verification tools [6]. Other research explores LLMs as

optimization agents, including LEO [7], and investigates broader LLM–optimization interaction [8]. However, most evaluations emphasize toy planning or synthetic tasks rather than classical OR benchmarks with known optimal objectives.

2.1.4 Identified Research Gap

Across the literature:

- UFLP research emphasizes solution algorithms, not automated modeling.
- ML for optimization improves heuristics, not symbolic formulation.
- LLM planning lacks systematic evaluation on OR-Library UFLP with exact optimality benchmarking.

To our knowledge, no work evaluates an LLM-generated MILP modeling pipeline on OR-Library UFLP instances with strict objective verification. This project addresses that gap by benchmarking LLM-generated symbolic models against OR-Library ground truth.

2.2 Baseline Model Specification

2.2.1 Deterministic Baseline: OR-Library Parser + OR-Tools MILP (CBC)

Baseline objective:

Implement the canonical UFLP MILP formulation and validate it on OR-Library instances using the published optimal objective values in uncapopt.txt [1]. This deterministic baseline acts as a ground-truth reference to verify correctness of parsing and modeling, and compute exact optimality gaps in later experiments..

Implementation components (reproducible notebook baseline):

1. Instance ingestion (OR-Library parsing): The notebook parses OR-Library uncapacitated warehouse location instances by extracting the fixed opening costs f_i (capacity ignored) and the assignment costs c_{ij} (demand ignored). This ensures consistent construction of the UFLP input (f, c)
2. Optimal value lookup (uncapopt.txt): A small parser loads uncapopt.txt and maps each instance name (e.g., cap71) to its known optimal objective value, enabling exact gap computation.
3. Canonical MILP formulation in OR-Tools (CBC backend).Decision variables:
    - $y_i \epsilon \{0,1\}$: facility opening decision
    - $x_{ij} \epsilon \{0,1\}$: customer assignment decision

Validation results (notebook execution):

We validated the baseline on three OR-Library instances of increasing facility count. The solver matched the known optimal objective values with 0.0% gap:

Instance	m	n	Solver objective	Best known	Gap (%)	Runtime (s)
cap71	16	50	932615.750	932615.750	0.0	0.057
cap101	25	50	796648.438	796648.437	0.0	0.084
cap131	50	50	793439.563	793439.562	0.0	0.196

Table 1: Deterministic baseline validation on OR-Library UFLP instances. The OR-Tools (CBC) MILP baseline matches the published best-known optimal objective values in uncapopt.txt with 0.0% optimality gap across increasing facility counts.

These results confirm:

- correct OR-Library parsing,
- correct MILP formulation,
- reliable solver configuration,
- a sound benchmark evaluation protocol using OR-Library ground truth [1].

Baseline vs "trained binary + retraining notebook" requirement:

Because the baseline is a deterministic optimization solver rather than a learned statistical model, we interpret "binary + retraining notebook" in terms of reproducible execution: the notebook regenerates all baseline results directly from raw OR-Library data, providing transparent, deterministic verification instead of retraining.

2.2.2 Proposed LLM-Augmented Model

The proposed system replaces manual model construction with LLM-generated symbolic optimization code while preserving solver-level verification. The pipeline is:

1. Parse structured instance data from OR-Library
2. Provide a constrained prompt template specifying UFLP structure and output format
3. LLM generates OR-Tools compatible MILP code
4. Execute generated code automatically
5. Compare objective against OR-Library optimum to compute exact gap and feasibility metrics

Planned extension: a solver-in-the-loop repair mechanism where solver errors or constraint violations are returned to the LLM for correction, consistent with formal verification paradigms [2], [6].

2.2.3 Metrics for Business and Research Evaluation

Evaluation must capture both technical optimization performance and business transformation impact.

Technical Metrics

- Optimality Gap (%): (Solver Objective−Best Known)/Best Known
- Feasibility Rate: percentage of instances producing valid constraint-satisfying solutions.
- Code Execution Success Rate: percentage of LLM-generated models that run without runtime errors.
- Constraint Satisfaction Rate: post-solve verification of all constraints.
- Solver Runtime: wall-clock time in seconds.

Business-Oriented Metrics

- Modeling Time Reduction: minutes from instance load to first feasible solution (manual vs LLM-generated).
- Manual Intervention Reduction: number of prompt revisions or code edits required to achieve feasibility.
- Scenario Iteration Speed: time required to modify parameters and regenerate a valid model.
- Scalability Across Instances: consistency of model generation across increasing instance sizes.

Research Metrics

- Generalization Across Instances: performance on unseen OR-Library instances.
- Prompt Stability: variation in gap or feasibility across multiple runs with identical prompts.
- Reproducibility: ability to regenerate identical results using the provided notebook and raw data.

The availability of exact optimal objective values in OR-Library enables strict quantitative benchmarking, strengthening the scientific validity of the study.

Works Cited:

[1] J. E. Beasley, “OR-Library: Distributing test problems by electronic mail,” J. Oper. Res. Soc., vol. 41, no. 11, pp. 1069–1072, 1990.

[2] Y. Hao, Y. Zhang, and C. Fan, “Planning Anything with Rigor: General-Purpose Zero-Shot Planning with LLM-based Formalized Programming,” arXiv preprint arXiv:2410.12112, 2024.

[3] A. Sonuç and E. Özcan, “ABPEA: Adaptive Binary Parallel Evolutionary Algorithm for the Uncapacitated Facility Location Problem,” Expert Systems with Applications, 2023.

[4] A. Kısaboyun and A. Sonuç, “A Hybrid Genetic Algorithm with Simulated Annealing for the Uncapacitated Facility Location Problem,” Mathematics, 2025.

[6] Y. Hao, Y. Zhang, and C. Fan, “Large Language Models Can Solve Real-World Planning Rigorously with Formal Verification Tools,” ICLR, 2025.

[7] X. Liu, Y. Wang, Z. Li, and J. Tang, “LEO: Large Language Models as Evolutionary Optimizers,” arXiv preprint arXiv:2403.02054, 2024.

[8] H. Huang, Z. Yang, Y. Chen, and C. Zhang, “When Large Language Model Meets Optimization,” arXiv preprint arXiv:2404.11891, 2024.

[9] J. Byrka, K. Fleszar, B. Rybicki, and J. Spoerhase, “Recent Advances in Approximation Algorithms for Facility Location Problems,” Mathematics, vol. 13, no. 10, 2025.