In the Name of God, the Most Gracious, the Most Merciful

MetaMind: LLM-Orchestrated Computational Intelligence Framework

Project Document

Course: Computational Intelligence **Instructor:** Dr.Mozayeni
Project Designer: Mohammad Sadegh Poulaei **Date:** December 2025

---

1. Project Overview

1.1 Objective

Students will develop an LLM-based intelligent agent that orchestrates various Computational Intelligence (CI) methods to solve optimization, classification, and clustering problems. The LLM analyzes the problem, selects appropriate method(s), configures parameters, executes the solution, evaluates results, and provides feedback for improvement.

1.2 Scope

- Implement 9 CI methods with standardized interfaces
- Develop LLM orchestrator for method selection and configuration
- Solve 4 benchmark problems across different domains
- Conduct experiments and statistical analysis
- Generate comprehensive results report

---

2. End-to-End System Flow

2.1 Complete Pipeline

Step 1: Problem Input to LLM

The user provides a problem description to the LLM. The input includes: - Problem type (optimization, classification, clustering) - Problem data (distance matrix, dataset, function definition) - Constraints and requirements - Performance expectations (speed vs accuracy preference)

Example input for TSP:

```
Problem: Traveling Salesman Problem
Cities: 30
Distance Matrix: [provided as file or inline]
Objective: Minimize total tour distance
Time Limit: 60 seconds
Priority: Solution quality over speed
```

Step 2: LLM Analysis and Method Selection

The LLM processes the input and performs: 1. Problem classification (combinatorial, continuous, supervised, unsupervised) 2. Complexity assessment (size, constraints, search space) 3. Method compatibility analysis 4. Selection of primary method with justification 5. Parameter configuration based on problem characteristics 6. Optional: suggest backup methods

LLM output format:

```
{
    "problem_type": "combinatorial_optimization",
    "selected_method": "ACO",
    "reasoning": "ACO is naturally suited for graph-based routing problems...",
    "parameters": {
        "n_ants": 30,
        "alpha": 1.0,
        "beta": 2.0,
        "evaporation_rate": 0.5,
        "iterations": 500
    },
    "backup_method": "GA",
    "confidence": 0.85
}
```

Step 3: Method Execution

The system: 1. Parses LLM output to extract method and parameters 2. Initializes the selected CI method with given parameters 3. Loads and preprocesses problem data 4. Executes the method 5. Records execution metrics (time, iterations, convergence history) 6. Stores the solution

Step 4: Evaluation

The system evaluates results using problem-specific metrics: - Computes fitness/accuracy/error values - Generates convergence curves - Compares with known optima (if available) - Calculates statistical measures across multiple runs

Step 5: Feedback to LLM

Results are sent back to LLM for interpretation:

```
{
    "method_used": "ACO",
    "best_solution": [0, 5, 2, 8, ...],
    "best_fitness": 427.3,
    "known_optimal": 420.0,
    "gap_percentage": 1.74,
    "computation_time": 45.2,
    "convergence_history": [850.2, 723.1, 612.4, ...],
    "iterations_completed": 500
}
```

Step 6: LLM Interpretation and Recommendations

The LLM analyzes results and provides: 1. Performance assessment (good/acceptable/poor) 2. Comparison with expected performance 3. Explanation of results in natural language 4. Recommendations for improvement: - Parameter tuning suggestions - Alternative method recommendations - Hybrid approach suggestions

LLM feedback output:

```
## Results Analysis

The ACO achieved a tour length of 427.3, which is 1.74% above the known
optimal of 420.0. This is considered GOOD performance for a metaheuristic
approach.

### Observations:
- Convergence was smooth, indicating stable search
- Solution found at iteration 412, suggesting adequate iteration count
- Computation time of 45.2s is within acceptable limits

### Recommendations:
1. Try increasing beta to 2.5 to strengthen greedy heuristic
2. Consider 2-opt local search for solution refinement
3. If time permits, run GA for comparison

### Confidence in solution: HIGH (gap < 2%)
```

Step 7: Iterative Improvement (Optional)

Based on LLM recommendations: 1. User can request another run with suggested parameters 2. System can automatically try backup method 3. Results are accumulated for comparative analysis

---

3. Features to Implement

3.1 LLM Orchestrator Features

Feature	Description	Priority
Problem Parser	Extract problem type, size, constraints from user input	Required
Method Selector	Choose appropriate CI method based on problem analysis	Required
Parameter Configurator	Set method parameters based on problem characteristics	Required
Result Interpreter	Analyze execution results and generate insights	Required
Recommendation Engine	Suggest improvements and alternative approaches	Required
Conversation Memory	Remember previous attempts for the same problem	Optional
Multi-method Orchestration	Run multiple methods and compare	Optional

3.2 Method Interface Features

Feature	Description	Priority
Standardized API	Common interface for all methods	Required
Parameter Validation	Validate parameters before execution	Required
Progress Callback	Report progress during execution	Required
Early Stopping	Stop when convergence detected	Optional
Checkpointing	Save/resume long runs	Optional
Logging	Detailed execution logs	Required

3.3 Evaluation Features

Feature	Description	Priority
Metric Computation	Calculate problem-specific metrics	Required
Statistical Analysis	Mean, std, confidence intervals	Required
Convergence Plotting	Visualize optimization progress	Required
Comparison Tables	Compare methods side by side	Required
Export Results	Save results in CSV/JSON format	Required

---

4. Methods to Support

4.1 Neural Network Methods

Perceptron

```
Parameters:
    - learning_rate: float (default: 0.01, range: 0.001-0.1)
    - max_epochs: int (default: 100, range: 50-1000)
    - bias: bool (default: True)
```

Multi-Layer Perceptron (MLP)

```
Parameters:
    - hidden_layers: list[int] (default: [64, 32], example: [128, 64, 32])
    - activation: str (default: "relu", options: "relu", "sigmoid", "tanh")
    - learning_rate: float (default: 0.001, range: 0.0001-0.01)
    - max_epochs: int (default: 500, range: 100-2000)
    - batch_size: int (default: 32, range: 16-128)
    - optimizer: str (default: "adam", options: "adam", "sgd", "rmsprop")
```

Kohonen Self-Organizing Map (SOM)

```
Parameters:
    - map_size: tuple (default: (10, 10), range: (5,5) to (50,50))
    - learning_rate_initial: float (default: 0.5, range: 0.1-1.0)
    - learning_rate_final: float (default: 0.01)
    - neighborhood_initial: float (default: 5.0)
    - max_epochs: int (default: 1000, range: 500-5000)
    - topology: str (default: "rectangular", options: "rectangular", "hexagonal")
```

Hopfield Network

```
Parameters:
    - max_iterations: int (default: 100, range: 50-500)
    - threshold: float (default: 0.0)
    - async_update: bool (default: True)
    - energy_threshold: float (default: 1e-6)
```

4.2 Fuzzy System

Fuzzy Controller

```
Parameters:
    - n_membership_functions: int (default: 3, options: 3, 5, 7)
    - membership_type: str (default: "triangular", options: "triangular", "gaussian", "trape
    - defuzzification: str (default: "centroid", options: "centroid", "bisector", "mom", "so
    - rule_generation: str (default: "wang_mendel", options: "wang_mendel", "manual")
```

4.3 Evolutionary Algorithms

Genetic Algorithm (GA)

```
Parameters:
    - population_size: int (default: 100, range: 50-500)
    - generations: int (default: 500, range: 100-2000)
    - crossover_rate: float (default: 0.8, range: 0.6-0.95)
    - mutation_rate: float (default: 0.1, range: 0.01-0.3)
    - selection: str (default: "tournament", options: "tournament", "roulette", "rank")
    - tournament_size: int (default: 3, range: 2-10)
    - elitism: int (default: 2, range: 0-10)
    - crossover_type: str (default: "pmx", options: "pmx", "ox", "cx" for permutation; "sing
```

Genetic Programming (GP)

```
Parameters:
    - population_size: int (default: 200, range: 100-1000)
    - generations: int (default: 50, range: 20-200)
    - max_depth: int (default: 6, range: 3-10)
    - crossover_rate: float (default: 0.9, range: 0.7-0.95)
    - mutation_rate: float (default: 0.1, range: 0.05-0.2)
    - function_set: list (default: ["+", "-", "*", "/"], options include: "sin", "cos", "exp
    - terminal_set: list (default: ["x", "constants"])
    - parsimony_coefficient: float (default: 0.001, range: 0-0.01)
```

Particle Swarm Optimization (PSO)

```
Parameters:
    - n_particles: int (default: 50, range: 20-200)
    - max_iterations: int (default: 500, range: 100-2000)
    - w: float (default: 0.7, range: 0.4-0.9) # inertia weight
    - c1: float (default: 1.5, range: 1.0-2.5) # cognitive coefficient
    - c2: float (default: 1.5, range: 1.0-2.5) # social coefficient
    - w_decay: bool (default: True) # linearly decrease w
    - velocity_clamp: float (default: 0.5, range: 0.1-1.0) # fraction of search range
```

Ant Colony Optimization (ACO)

```
Parameters:
    - n_ants: int (default: 50, range: 10-100)
    - max_iterations: int (default: 500, range: 100-2000)
    - alpha: float (default: 1.0, range: 0.5-2.0) # pheromone importance
    - beta: float (default: 2.0, range: 1.0-5.0) # heuristic importance
    - evaporation_rate: float (default: 0.5, range: 0.1-0.9)
    - q: float (default: 1.0) # pheromone deposit factor
    - initial_pheromone: float (default: 0.1)
    - local_search: bool (default: True) # apply 2-opt improvement
```

---

5. Problem Specifications and Evaluation Metrics

5.1 Problem 1: Traveling Salesman Problem (TSP)

Description

Find the shortest route that visits each city exactly once and returns to the starting city.

Test Instances

Instance	Cities	Source	Known Optimal
eil51	51	TSPLIB	426
berlin52	52	TSPLIB	7542
kroA100	100	TSPLIB	21282
Random30	30	Generated	Compute via exact solver
Random50	50	Generated	Estimate via LKH

Evaluation Metrics

Metric	Formula	Description
Tour Length	Sum of edge distances	Primary objective (minimize)
Gap to Optimal	(found - optimal) / optimal × 100%	Quality measure
Computation Time	Seconds elapsed	Efficiency measure
Success Rate	% runs within 5% of optimal	Reliability measure
Convergence Speed	Iterations to reach 90% of final quality	Search efficiency

---

5.2 Problem 2: Function Optimization

Description

Find the global minimum of multimodal benchmark functions with known optima.

Test Functions

Rastrigin Function

```
f(x) = 10n + Σ [x ² - 10cos(2 x )]
Domain: x   [-5.12, 5.12]
Global minimum: f(0, 0, ..., 0) = 0
Dimensions to test: n = 10, 20, 30
Characteristics: Highly multimodal, regular structure
```

Ackley Function

```
f(x) = -20exp(-0.2√(1/n Σx ²)) - exp(1/n Σcos(2 x )) + 20 + e
Domain: x   [-5, 5]
Global minimum: f(0, 0, ..., 0) = 0
Dimensions to test: n = 10, 20, 30
Characteristics: Large flat region, deep hole at optimum
```

Rosenbrock Function

```
f(x) = Σ [100(x   - x ²)² + (1-x )²]
Domain: x   [-5, 10]
Global minimum: f(1, 1, ..., 1) = 0
Dimensions to test: n = 10, 20, 30
Characteristics: Narrow curved valley, difficult convergence
```

Sphere Function (baseline - easy)

```
f(x) = Σ  x ²
Domain: x   [-5.12, 5.12]
Global minimum: f(0, 0, ..., 0) = 0
Dimensions to test: n = 10, 20, 30
Characteristics: Unimodal, smooth, for baseline comparison
```

Evaluation Metrics

Metric	Formula	Description
Best Fitness	f(x*)	Best value found
Mean Fitness	Average of best fitness across runs	Consistency
Std Dev	Standard deviation of best fitness	Reliability
Error	|f(x*) - f(x_opt)|	Absolute error
Success Rate	% runs with error < 1e-4	Reliability
Function Evaluations	Total calls to objective function	Efficiency

---

5.3 Problem 3: Classification (Titanic Dataset)

Description

Predict passenger survival on the Titanic based on features like age, sex, class, etc.

Dataset Specifications

Property	Value
Source	Kaggle Titanic Dataset
Samples	891 training, 418 test

Property	Value
Features	11 (after preprocessing: ~8-10)
Classes	2 (Survived: 0 or 1)
Class Balance	~38% survived, ~62% died

Features

```
- Pclass: Passenger class (1, 2, 3)
- Sex: Male/Female (encode as 0/1)
- Age: Age in years (handle missing values)
- SibSp: Number of siblings/spouses aboard
- Parch: Number of parents/children aboard
- Fare: Ticket fare
- Embarked: Port of embarkation (C, Q, S)
- Cabin: Cabin number (high missing rate, may drop or engineer)
```

Preprocessing Requirements - Handle missing values (Age, Cabin, Embarked) - Encode categorical variables - Normalize/standardize numerical features - Train/validation/test split (e.g., 70/15/15)

Evaluation Metrics

Metric	Formula	Description
Accuracy	(TP + TN) / Total	Overall correctness
Precision	TP / (TP + FP)	Positive predictive value
Recall	TP / (TP + FN)	Sensitivity
F1 Score	2 × (Precision × Recall) / (Precision + Recall)	Balanced measure
AUC-ROC	Area under ROC curve	Ranking quality
Confusion Matrix	[[TN, FP], [FN, TP]]	Detailed breakdown
Cross-Validation Score	Mean accuracy across k folds	Generalization

---

5.4 Problem 4: Clustering

Description

Discover natural groupings in unlabeled data.

Datasets

Dataset A: Iris (for validation)

```
Samples: 150
Features: 4 (sepal length/width, petal length/width)
True clusters: 3 (species: setosa, versicolor, virginica)
Use: Validate clustering against known labels
```

Dataset B: Mall Customers

```
Source: Kaggle Mall Customer Segmentation
Samples: 200
Features: 5 (CustomerID, Gender, Age, Annual Income, Spending Score)
Use features: Age, Annual Income, Spending Score
Expected clusters: 4-6 customer segments
```

Dataset C: Synthetic Data

```
Generate using sklearn.datasets.make_blobs:
- n_samples: 500
- n_features: 2, 5, 10 (test different dimensions)
- n_clusters: 5
- cluster_std: 1.0
Use: Controlled environment for parameter tuning
```

Evaluation Metrics

Metric	Description	When to Use
Silhouette Score	Cohesion vs separation (-1 to 1, higher is better)	Always
Davies-Bouldin Index	Cluster similarity (lower is better)	Always
Calinski-Harabasz Index	Variance ratio (higher is better)	Always
Adjusted Rand Index	Agreement with true labels (0 to 1)	When true labels available
Normalized Mutual Information	Information shared with true labels	When true labels available
Inertia	Within-cluster sum of squares	For comparison

---

6. Experimental Protocol

6.1 LLM Orchestrator Evaluation

1. Present each problem to LLM orchestrator 5 times
2. Record: selected method, reasoning, suggested parameters
3. Execute selected method with suggested parameters
4. Compare results with best fixed-method performance
5. Evaluate LLM selection accuracy and parameter quality

---

7. Expected Results Report

7.1 Report Structure

Section 1: Introduction (1-2 pages) - Project objectives - System overview - Methods and problems summary

Section 2: Implementation Details (3-5 pages) - System architecture - Method implementations (brief description of each) - LLM integration approach - Challenges and solutions

Section 3: Experimental Setup (2-3 pages) - Hardware and software environment - Parameter settings for all methods - Evaluation metrics definitions - Statistical testing methodology

Section 4: Results per Problem (8-12 pages)

For each problem, include:

4.x.1 Problem Description - Problem specifics - Test instances used

4.x.2 Results Table

Example for TSP:

Method	Best	Mean ± Std	Time (s)	Success Rate	Rank
ACO	428.2	435.1 ± 5.2	34.5	73%	1
GA	432.1	441.3 ± 7.8	28.3	60%	2
PSO	445.6	459.2 ± 12.1	22.1	43%	3
Hopfield	512.3	534.2 ± 28.4	5.2	10%	4

4.x.3 Convergence Curves - Plot showing fitness vs iterations for each method - Average curve with confidence bands

4.x.4 Statistical Analysis - Wilcoxon test results (p-values) - Significant differences highlighted

4.x.5 LLM Orchestrator Performance - Methods selected by LLM - Comparison with best fixed method - Quality of parameter suggestions

Section 5: LLM Orchestrator Analysis (3-4 pages) - Selection accuracy per problem type - Quality of reasoning - Parameter suggestion effectiveness - Improvement recommendations quality - Failure cases analysis

Section 6: Discussion (2-3 pages) - Best methods per problem type summary - LLM orchestrator effectiveness - Limitations observed - Lessons learned

Section 7: Conclusion (1 page) - Key findings - Project achievements - Future improvements

Appendices - A: Complete parameter settings - B: All experimental results (tables) - C: LLM prompts used - D: Sample LLM interactions

7.2 Summary Results Table (Expected)

Problem	Best Method	LLM Selected	LLM Accuracy
TSP (small)	ACO	ACO/GA	High
TSP (large)	GA	GA	High
Rastrigin	PSO	PSO	High
Ackley	PSO	PSO	High
Rosenbrock	GA	PSO/GA	Medium
Titanic	MLP	MLP	High
Clustering	Kohonen	Kohonen	High

---

8. Deliverables

8.1 Code Deliverables

Deliverable	Description	Format
Source Code	Complete implementation	Python package
Method Library	All 9 CI methods	src/methods/
LLM Orchestrator	Agent implementation	src/orchestrator/
Problem Implementations	All 4 problems	src/problems/
Experiment Scripts	Reproducible experiments	experiments/
Requirements	Dependencies	requirements.txt
README	Setup and usage instructions	README.md

8.2 Documentation Deliverables

Deliverable	Description	Format
Final Report	Complete analysis (20-30 pages)	PDF
User Guide	How to use the system	Markdown

8.3 Data Deliverables

Deliverable	Description	Format
Raw Results	All experimental runs	CSV/JSON
Processed Results	Statistical summaries	CSV
Figures	Convergence curves, comparisons	PNG/PDF
LLM Logs	All LLM interactions	JSON