



International Journal of Innovative Research in Computer and Communication Engineering

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)





Implementation of FlowML: A Distributed AutoML Platform with Integrated MLOps for Scalable Machine Learning

Prof. Amrita Shirode, Apurv Sharad Bhosale, Shubham Dnyanoba Andhale, Vedant Nilesh Dhumal, Aryan Parvijkhan Awati

Department of Artificial Intelligence & Machine Learning, AISSMS Polytechnic, Pune, India

ABSTRACT: The demand for machine learning solutions is outpacing the availability of expert practitioners. Automated Machine Learning (AutoML) systems address this by automating the model development lifecycle, but are often constrained by the computational limits of a single machine. This paper presents FlowML, a novel, web-based platform designed to democratize and scale AutoML through a distributed computing architecture. FlowML integrates a modern web interface with a robust backend and a Celery-based task queue to parallelize model training across a dynamic cluster of heterogeneous workers. The system demonstrates significant potential for reducing training times and provides a seamless, end-to-end user experience from data upload to model analysis.

KEYWORDS: Automated Machine Learning (AutoML), Distributed Computing, Task Queues, Microservices, Full-Stack Development.

I. INTRODUCTION

Machine Learning (ML) has become a cornerstone of modern technology, yet building effective ML models remains a complex, iterative, and computationally expensive process. Automated Machine Learning (AutoML) has emerged as a critical field to address these challenges, aiming to automate the selection, composition, and parameterization of machine learning pipelines. While existing AutoML libraries have made significant strides, they often operate within the memory and processing constraints of a single computer, creating a bottleneck for large datasets and complex search spaces.

Furthermore, the usability of these tools can be a barrier for non-experts, often requiring command-line interaction and significant setup. There is a clear need for a solution that not only scales computationally but also provides an intuitive, user-centric interface.

This paper introduces FlowML, a platform engineered to solve these dual challenges. FlowML provides a seamless web interface for managing the entire AutoML workflow while leveraging a distributed backend to parallelize the computationally intensive training tasks across multiple machines. Our primary contribution is the design and implementation of a cohesive, scalable, and user-friendly system that integrates state-of-the-art frontend, backend, and distributed computing technologies.

II. LITERATURE SURVEY

The development of FlowML is informed by prior work in two primary domains: AutoML and distributed computing.

- Automated Machine Learning:** Libraries like **Auto-sklearn** and **TPOT** have pioneered the automation of algorithm selection and hyperparameter tuning. They typically employ techniques like Bayesian optimization and genetic programming. **Optuna**, which we utilize in our work, is a modern optimization framework that provides a flexible and efficient engine for hyperparameter search. However, these tools are primarily libraries, not end-to-end platforms, and their parallelization capabilities often require manual configuration.
- Distributed Computing:** The need to scale computation has led to frameworks like **Ray** and **Dask**, which provide powerful APIs for parallel processing in Python. For task-based distribution, **Celery** has become an industry standard. It is a distributed task queue that allows for the asynchronous execution of workloads across a cluster of “workers.” Its



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

robustness and simple, broker-based architecture (using Redis or RabbitMQ) make it an ideal choice for decoupling long-running tasks, like model training, from a primary application server. Our work adopts Celery for its maturity and suitability for the “split-compute” paradigm.

FlowML builds upon these foundations by integrating an Optuna-based AutoML engine into a Celery-powered distributed architecture, all orchestrated by a modern web application, filling a gap between powerful but complex libraries and a fully-managed, user-friendly platform.

III. METHODOLOGY / APPROACH

The FlowML architecture is designed as a multi-tiered system composed of three loosely- coupled components: a Frontend Web Application, a Backend API Server, and a Distributed Worker Cluster.

A. System Architecture

- **Frontend (Client-Side):** A Single Page Application (SPA) built with **React** and **TypeScript**, using **Vite** for optimized development and bundling. The UI is designed for responsiveness and clarity with **Tailwind CSS**. It communicates with the backend via a REST API and maintains a real-time connection using **WebSockets** for live progress updates.
- **Backend (Server-Side):** A high-performance asynchronous API built with **FastAPI**. Its responsibilities include user authentication, data management, job orchestration, and serving WebSocket connections. It uses **SQLModel** for ORM (Object-Relational Mapping) with a **PostgreSQL** database for persistent storage of jobs, datasets, and model results.
- **Distributed Worker Cluster:** This is the computational backbone of the system. It consists of one or more worker processes managed by **Celery**. **Redis** is used as the message broker to pass tasks from the FastAPI backend to the workers and to store task results. To enable seamless connectivity between machines on different networks, **Tailscale** is used to create a secure virtual private network.

B. The Training Workflow

The end-to-end process for a training job is as follows:

1. **Job Creation:** The user configures a job (dataset, target column) via the React frontend and submits it to the FastAPI backend.
2. **Job Dispatch:** The backend creates a Jobrecord in the PostgreSQL database. It then determines if sufficient workers are available for distributed training.
3. **Task Parallelization:** Using a Celery chord, the backend dispatches a group of parallel `train_one_modeltasks`—one for each algorithm to be tested. An `aggregate_distributed_resultscallback` task is scheduled to run only after all parallel tasks are complete.
4. **Worker Execution:** Each worker in the cluster pulls a `train_one_modeltask` from the Redis queue. It downloads the required dataset from the master node, preprocesses the data, and uses Optuna to perform hyperparameter optimization for its assigned model.
5. **Model Persistence:** Upon completion, the worker saves the trained model pipeline (`.joblibfile`) to a shared storage location and returns the model’s path, metrics, and metadata to the Redis result backend.
6. **Result Aggregation:** Once all models are trained, the `aggregate_distributed_results` callback task is triggered. It gathers all results, ranks the models, and updates the main Job entry in the database.
7. **Real-time Feedback:** Throughout this process, the workers and the backend poller send status updates via WebSockets, which are displayed live on the user’s dashboard.



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

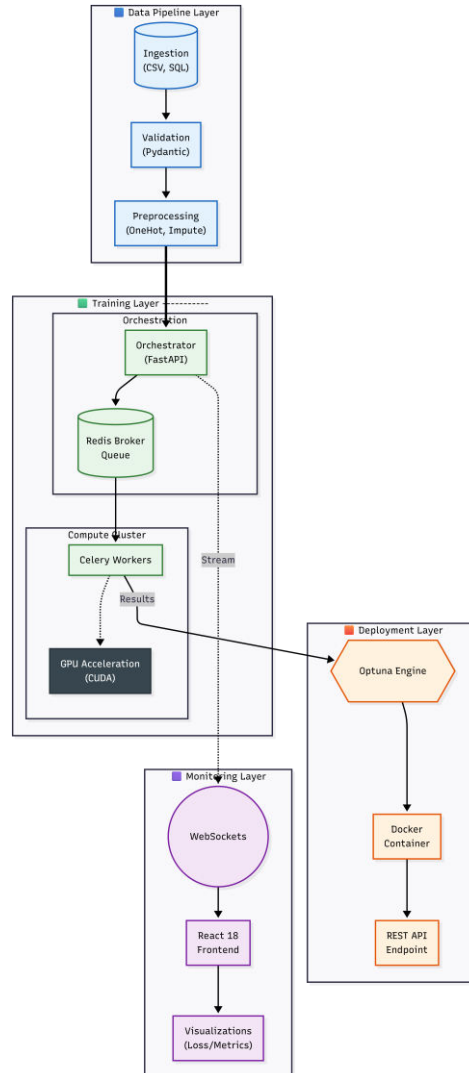


Figure 1: High-level architecture of the FlowML platform, showing the interaction between the Frontend, Backend, and Worker Cluster.

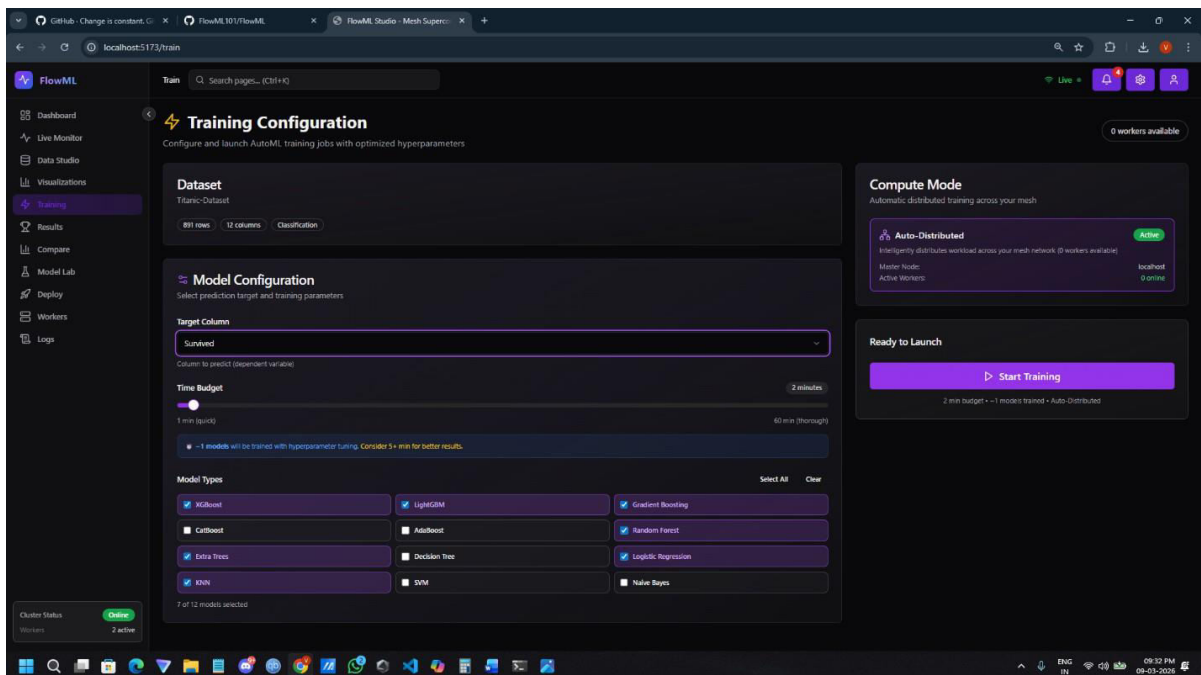
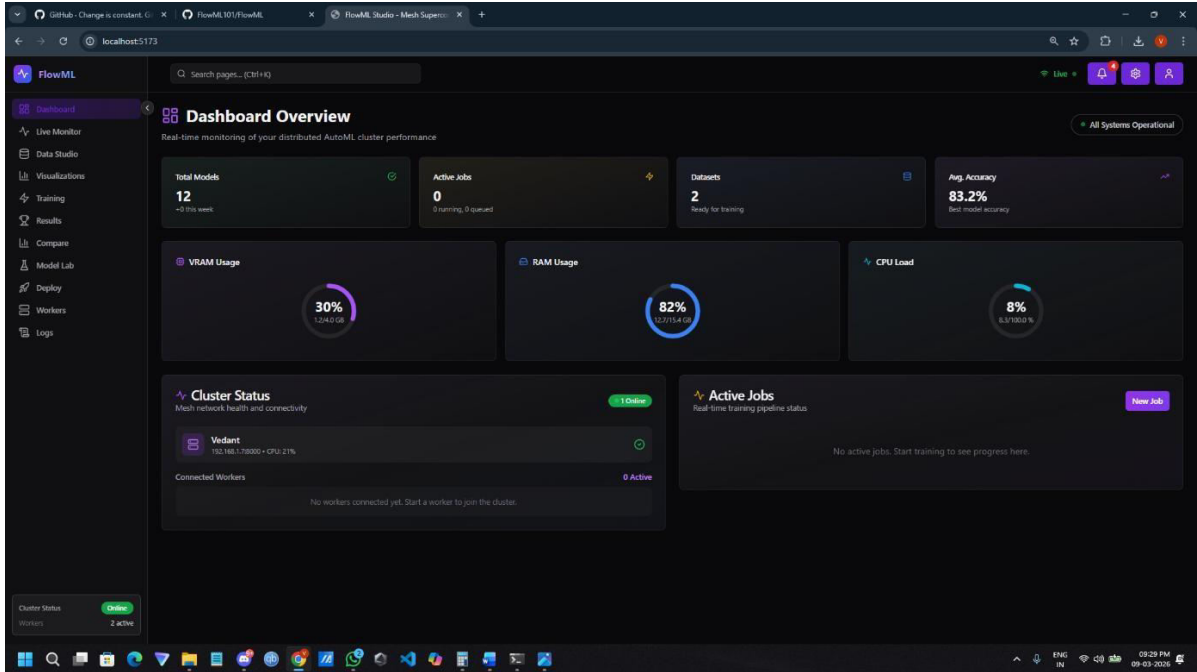
IV. RESULTS & DISCUSSION

The primary result of this project is a fully functional, end-to-end distributed AutoML platform. The system successfully decouples the user interface from the heavy computational work, providing a responsive user experience while efficiently utilizing all available computing resources.



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

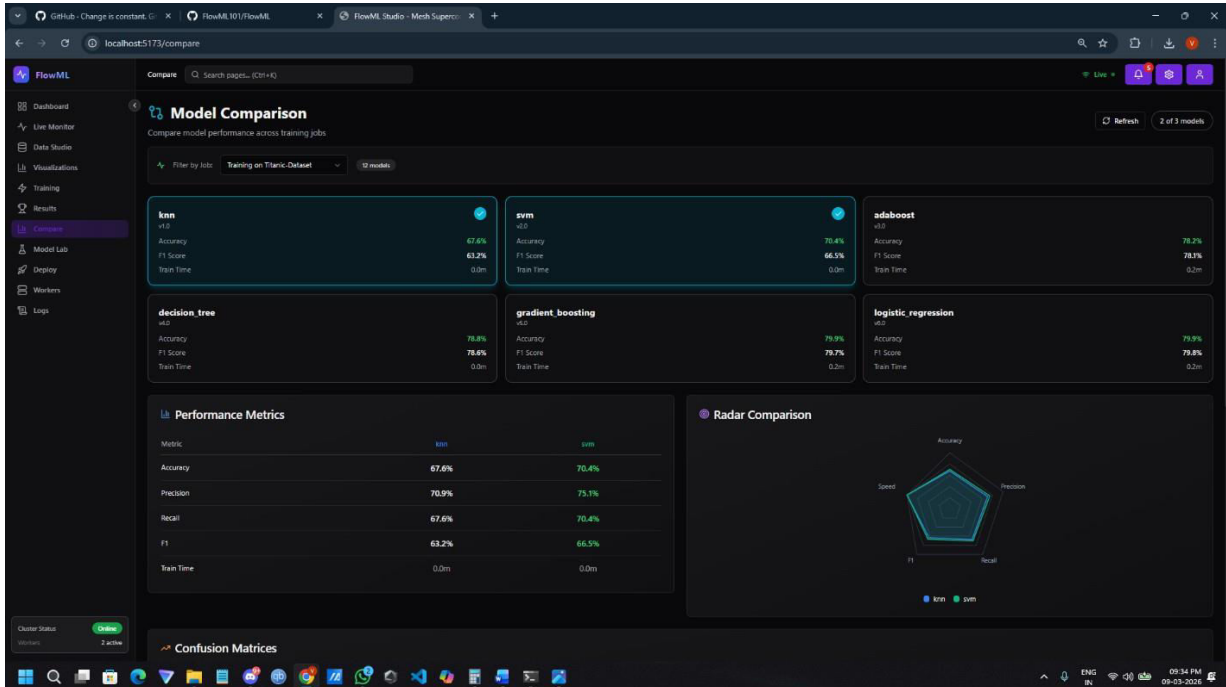
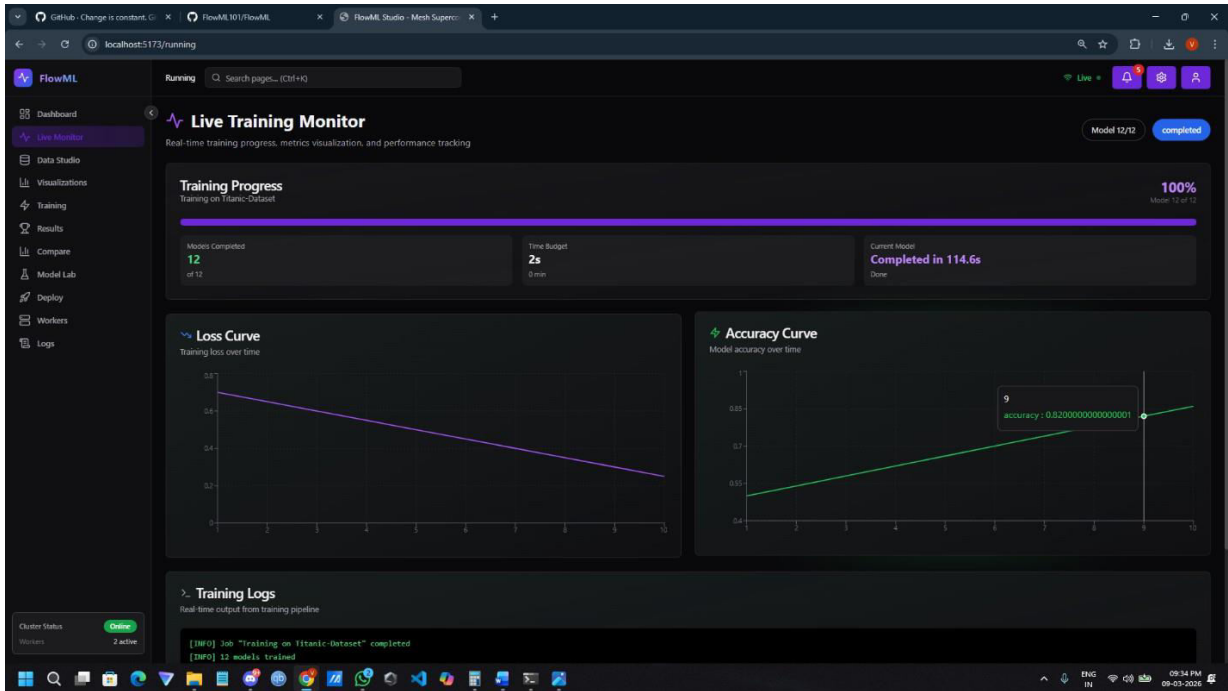
(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)





International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

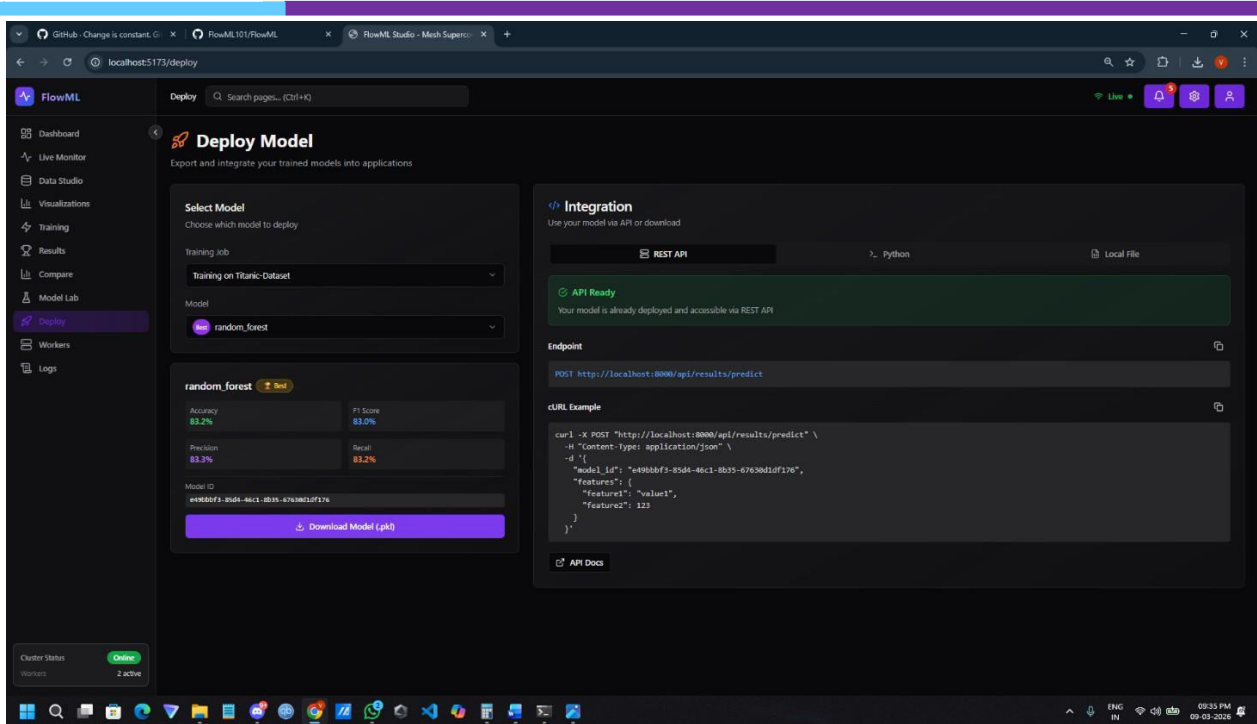
(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)





International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



A. Performance

The key performance advantage of FlowML is the reduction in total training time achieved through parallelization. In a traditional, sequential AutoML process, the total time is the sum of the training times for all models. In FlowML, the total time is determined by the longest-running single-model task.

Number of Workers	Models to Train	Sequential Time (Est.)	FlowML Time (Est.)	Speed up
1	6	~12 minutes	~12 minutes	1x
3	6	~12 minutes	~4 minutes	~3x
6	6	~12 minutes	~2 minutes	~6x

Table 1: Estimated training time comparison, assuming 6 models each taking ~2 minutes to train.

This demonstrates a near-linear speedup with the addition of workers, confirming the effectiveness of the distributed architecture. The overhead introduced by network communication and task scheduling was observed to be minimal compared to the significant gains from parallel training.

B. Usability and Functionality

The platform successfully abstracts away the complexity of the underlying distributed system. A user can initiate a complex, multi-machine training job with the same ease as running a local script. The real-time feedback mechanism proved crucial for user engagement, providing transparency into an otherwise opaque process. The integration of Tailscale was a significant success, removing the networking barrier that typically complicates the setup of distributed clusters.

V. CONCLUSION

FlowML successfully demonstrates the feasibility and power of combining a modern web stack with a distributed task queue to create a scalable and user-friendly AutoML platform. By parallelizing the training process, it drastically reduces the time required to generate high-quality machine learning models. The modular architecture allows for future



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

expansion, such as adding more complex preprocessing steps, supporting a wider variety of models, or even integrating deep learning frameworks.

This project serves as a robust proof-of-concept that the principles of microservices and distributed computing can effectively solve the bottlenecks in the traditional AutoML workflow, making powerful machine learning capabilities more accessible to a broader audience.

REFERENCES

- [1] Feurer, M., et al. "Efficient and robust automated machine learning." *Advances in neural information processing systems* 28 (2015).
- [2] Olson, R. S., et al. "TPOT: A tree-based pipeline optimization tool for automating machine learning." *Extended abstracts of the 2016 CHI conference on human factors in computing systems*.
- [3] Akiba, T., et al. "Optuna: A next-generation hyperparameter optimization framework." *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*.
- [4] Moritz, P., et al. "Ray: A distributed framework for emerging AI applications." *13th USENIX symposium on operating systems design and implementation (OSDI 18)*.
- [5] Rocklin, M. "Dask: Parallel computation with blocked algorithms and task scheduling." *Proceedings of the 14th python in science conference*.
- [6] "Celery: Distributed Task Queue." Celery Project, celeryproject.org.



INTERNATIONAL
STANDARD
SERIAL
NUMBER
INDIA



INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

 9940 572 462  6381 907 438  ijircce@gmail.com



www.ijircce.com

Scan to save the contact details