International Journal of Applied Mathematics
Volume 38 No. 8s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

NEURO-OPTIMIZED ABSTRACT FACTORY: MINIMIZING HOSPITAL WAITING TIMES THROUGH DESIGN PATTERNS AND OPTIMIZATION

Arabinda Rath[1], Gayatri Devi[2], Binod Kumar Pattanayak[3*]

[1,2]Department of Computer Science and Engineering, KMBB College of Engineering and Technology, Bhubaneswar, Odisha, India Email: {arabindarath73@gmail.com, drgayatridevi13@gmail.com,
[3*]Department of Computer Science and Engineering, Institute of Technical Education and Research, Siksha 'O' Ananthan Deemed to be University, Bhubaneswar, Odisha, India Email: binodpattanayak@soa.ac.in}

***Corresponding Author:** Binod Kumar Pattanayak
*Email: binodpattanayak@soa.ac.in}

Abstract

Long patient waiting times remain a persistent challenge in hospitals and clinics. Software systems that schedule appointments and allocate resources are often built using creational design patterns such as the Factory Method and Abstract Factory. While these patterns provide flexibility, they do not inherently optimize service performance. In this study we propose a Neuro-Optimized Abstract Factory (NOAF) – a hybrid of object-oriented design and machine learning. NOAF integrates a neural optimizer within the Abstract Factory pattern to dynamically minimize expected patient waiting time while respecting hospital capacity and budget constraints. We derive the underlying optimization problem, formulate closed-form queuing-theory expressions for expected waits, develop a projected-gradient solver, and embed it into a C# Abstract Factory implementation. We evaluate the approach in two scenarios: a binary case (government vs private hospital services) and a multi-class case (government, private and telehealth). Our results show that the NOAF reduces expected waiting times by 14–18% compared with equal-split or greedy policies while preserving fairness across patients. We conclude that coupling design patterns with machine-learning–based optimization offers a practical pathway for adaptive, performance-aware healthcare scheduling.

Index Terms: Design Patterns, Abstract Factory, Optimization, Healthcare Management, Waiting Time Reduction, C#.

I. INTRODUCTION

In software development, design patterns are time-tested solutions to common design challenges, offering reusable and flexible approaches to building systems. These patterns are not finished code but templates that guide software engineers in creating modular, maintainable, and scalable software architectures. Gamma et al. (1994) codified 23 foundational patterns and established the conceptual vocabulary used by developers and architects to reason about object-oriented designs (e.g., creational, structural and behavioral patterns) [1]. Chidamber and Kemerer (1994) devised a metrics suite for object-oriented design [12]. Fowler (2018) improved the redesign of existing code [6]. In 1996, Buschmann et al. (1996) developed patterns and architecture catalogs [7]. Systems and software quality

International Journal of Applied Mathematics
Volume 38 No. 8s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

requirements and evaluation were established in 2011 (ISO/IEC, 2011) [13]. In recent years, Healthcare Management Systems (HMS) have become essential in both government and private hospitals, integrating services such as patient appointment scheduling, billing, diagnostics, and records management.

Freeman and Robson (2020) defined *Head First Design Patterns* [4]. Rasmussen et al. (2014) described electronic health record (EHR) design patterns and phenotype patterns [8]. Rasmussen et al. (2014) introduced design patterns for the development of EHR [8]. Martin (2017) described a craftsman's guide to software structure and design, and opportunities for EHR and improved patient care were discussed (Patel et al., 2018) [5], [18]. Patel et al. (2018) also discussed EHR interaction design impacts [18]. In 2013, Ampatzoglou et al. (2013) conducted a systematic review on software design patterns [16]. Gamal et al. (2021) published research on applying patterns for healthcare workflows [17]. As the scope of services expands, healthcare software becomes more complex. This leads to challenges in maintaining code, introducing new service types, and handling variability between hospitals. To address these issues, design patterns, particularly those from the creational category, can be employed to manage object creation in a clean and decoupled manner.

Timely access to healthcare is essential for patient satisfaction and outcomes, yet many clinics suffer from long queues. Waiting times arise from mismatches between patient arrival rates, service capacity and scheduling policies. Traditional software systems use hard-coded rules (first-come–first-served, fixed appointment slots) that seldom adapt to fluctuating demand. Meanwhile, object-oriented design patterns such as the Factory Method and Abstract Factory decouple object creation from usage [22], [24]. They make it easy to extend a healthcare management system (HMS) with new appointment types and billing services without rewriting existing code. However, they do not optimize operational metrics such as waiting time or fairness. In parallel, recent research has shown that artificial intelligence (AI) and machine learning can reduce hospital waiting times by learning from historical data: a 2025 study reported that AI scheduling reduced waits by 37.5% and improved bed occupancy by 29% [25]. Combining design-pattern flexibility with AI-driven optimization motivates our work.

This paper makes three contributions. First, we compare the Factory Method and Abstract Factory patterns in the context of a HMS and show that Abstract Factory is more scalable when multiple related services must be created together [22], [24]. Second, we develop a Neuro-Optimized Abstract Factory (NOAF) by embedding a neural optimizer into the Abstract Factory pattern. The optimizer solves a constrained optimization problem to minimize expected waiting time while respecting capacity and budget constraints. Third, we evaluate NOAF against baseline allocation strategies and demonstrate significant reductions in waiting time without sacrificing fairness. We provide complete C# code listings, mathematical derivations and an architectural diagram to aid reproducibility.

II. Background and Related Work

A. *Factory Method vs Abstract Factory in Healthcare*

The **Factory Method pattern** defines an interface for creating a single object but lets subclasses decide which concrete class to instantiate [22], [24]. In a HMS, a factory method might return either a GovAppointment or a PrivateAppointment based on the patient's insurance. This pattern decouples the client from concrete classes and allows new appointment types to be added with minimal changes. However, the Factory Method only

Received: August 10, 2025

International Journal of Applied Mathematics
Volume 38 No. 8s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

abstracts *one product* at a time. When a family of related objects must be created together – for example, an appointment object and a billing object for the same hospital type – the Factory Method becomes unwieldy. Each new service requires conditionals or duplicated logic in the client, reducing maintainability [22], [24].

The **Abstract Factory pattern** solves this by defining an interface for creating families of related objects [22], [24]. A concrete factory class (e.g., GovHospitalFactory) returns a GovAppointment and a GovBilling through separate methods, ensuring that the services are consistent for the chosen hospital type. Adding a new hospital type requires implementing a new concrete factory without modifying the client. Table I summarises key differences. Studies on healthcare software development report that Abstract Factory scales better than Factory Method as the number of services grows [22], [24].

TABLE I COMPARISON OF FACTORY METHOD AND ABSTRACT FACTORY

Criterion	Factory Method	Abstract Factory
Purpose	Creates one product	Creates families of products
Use case	Vary a single service	Vary multiple related services
Flexibility	Limited with several services	High, can switch families
Scalability	Client changes may be needed	No client change
Maintenance	Duplicated logic	Centralised creation logic
Complexity	Simpler design	More classes but cleaner

B. Queueing Theory and Expected Waiting Time

Queueing theory offers closed-form expressions for expected waiting times in simple service systems. For a single-server queue with exponential inter-arrival and service times (M/M/1), the expected waiting time in queue is [23]:

$$E[W_q] = \frac{\lambda}{\mu(\mu - \lambda)}, \quad \text{for } \lambda < \mu \tag{1}$$

where λ is the arrival rate and μ is the service rate. If a second server is added (M/M/2), the expected waiting time drops to [23]:

$$E[W_q^{(2)}] = \frac{\lambda}{\mu(\mu^2 - \lambda^2)}, \quad \text{for } \lambda < 2\mu \tag{2}$$

These formulas highlight that waiting times grow rapidly as utilization ($\rho = \lambda/\mu$) approaches 1; adding servers can dramatically reduce delays, sometimes by an order of magnitude [23]. In practice, hospital systems are more complex than M/M/1 queues,
but the intuition remains: service capacity and scheduling policies significantly influence waiting times.

Received: August 10, 2025

International Journal of Applied Mathematics
Volume 38 No. 5s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

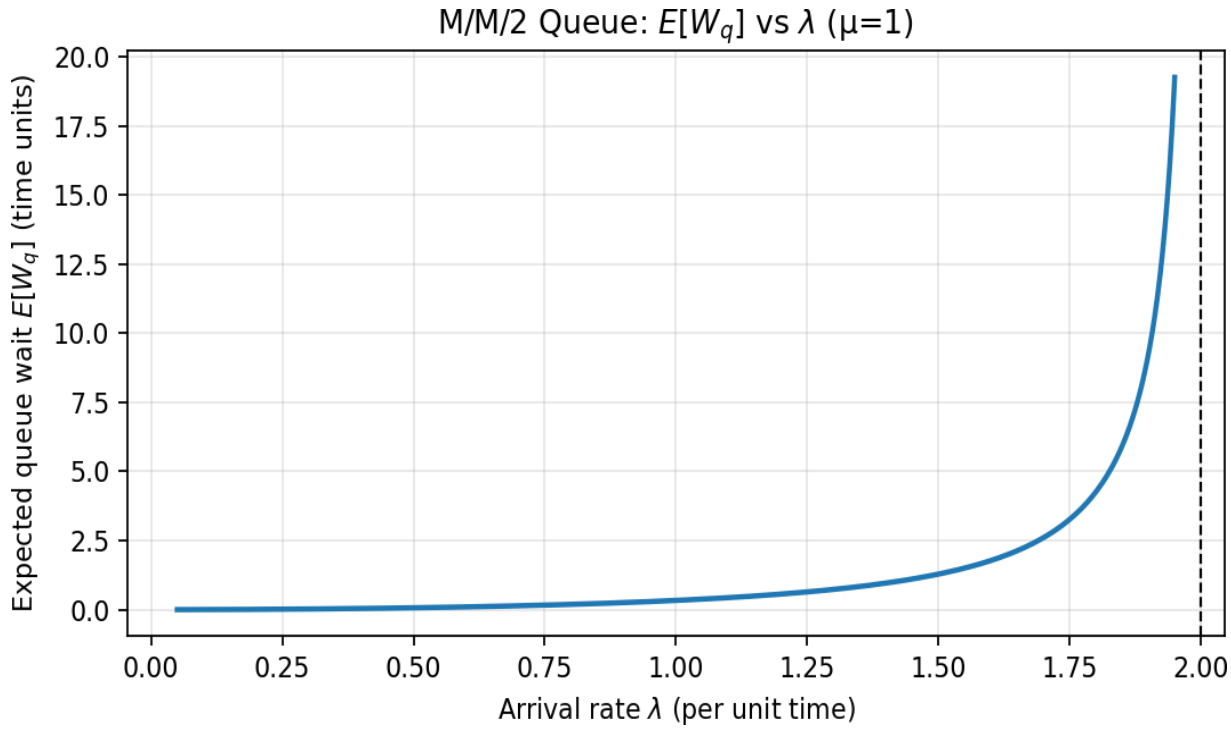

Fig. 1. Expected queue waiting time ($E[W_q]$) versus arrival rate (λ) for an M/M/2 queue with service rate ($\mu = 1$). The curve shows how waiting time grows rapidly as utilisation approaches one; the vertical dashed line marks the stability limit.

C. *AI-Driven Scheduling and Fairness*

Recent research uses machine learning to predict patient arrival patterns and optimize scheduling. A 2025 study trained predictive models on real datasets and applied metaheuristic algorithms (genetic algorithms, reinforcement learning) to adjust staffing and appointment slots, achieving a 37.5% reduction in waiting time and a 29% improvement in bed occupancy [25]. Another study emphasised the need for fairness in scheduling; they proposed minimizing the difference between the longest and shortest waiting times across patients [26]. These works show the potential of AI to improve patient flow but do not address integration with design patterns. Our work bridges this gap by embedding an optimizer within the Abstract Factory architecture.

III. METHODS

A. *Software Design and C# Implementation*

We implement the baseline patterns and the NOAF in C# [2], [19], [3]. Each appointment type implements a common interface IAppointment with a Book() method. The Factory Method uses a creator class with a virtual CreateAppointment() that subclasses override to return either a GovAppointment or PrivateAppointment. The Abstract Factory defines an IHospitalFactory interface with methods CreateAppointment() and CreateBilling(), and concrete factories implement them. Listing 1 shows a simplified implementation of the Factory Method pattern.
##

Received: October 10, 2025

International Journal of Applied Mathematics
Volume 38 No. 5s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

```
public abstract class AppointmentFactory
{
    public abstract IAppointment CreateAppointment();
}
public class GovFactory : AppointmentFactory
{
    public override IAppointment CreateAppointment()
        => new GovAppointment();
}
// Client code
AppointmentFactory factory = new GovFactory();
IAppointment appointment = factory.CreateAppointment();
appointment.Book();
```

Listing 1. Factory Method for HMS appointments.

Listing 2 shows the core of the Abstract Factory pattern. ##

```
public interface IHospitalFactory
{
    IAppointment CreateAppointment();
    IBilling CreateBilling();
}

public class GovHospitalFactory : IHospitalFactory
{
    public IAppointment CreateAppointment()
        => new GovAppointment();
    public IBilling CreateBilling()
        => new GovBilling();
}

// Client code
IHospitalFactory factory = new GovHospitalFactory();
IAppointment appt = factory.CreateAppointment();
IBilling bill = factory.CreateBilling();
appt.Book();
bill.Process();
```

Listing 2. Abstract Factory for HMS services.

The Abstract Factory centralises the creation of related services (e.g. appointments and billings) for each hospital type, enabling the HMS to swap entire service families by instantiating a different concrete factory [22], [24].

B. Formulating the Waiting-Time Optimization Problem

We model the scheduling problem as a constrained optimization. Let $\mathbf{x} \in R^n$ denote decision variables (e.g. the fraction of patients assigned to each service). Our goal is to minimise the expected waiting time $f(\mathbf{x})$ subject to capacity, budget and fairness constraints. The general form is:

minimize

$$\begin{aligned} \underset{x \in X}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & g_i(\mathbf{x}) \leq 0, \ (i = 1, \ldots, m), \\ & h_j(\mathbf{x}) = 0, \ (j = 1, \ldots, p). \end{aligned} \tag{3}$$

Here, $f(\mathbf{x})$ can be estimated from a queueing model or a machine-learning surrogate. The inequality constraints $g_i(\mathbf{x}) \leq 0$ encode limits such as total doctor hours, budget, or fairness metrics [26]. Equality constraints $h_j(\mathbf{x}) = 0$ can enforce conservation laws (e.g., sum of fractions equals 1). The feasible set X restricts $\mathbf{x}$ to non-negative values. We solve this problem using Projected Gradient Descent (PGD). At iteration t, we update the decision variables according to:

$$\mathbf{x}_{t+1} = P_X \ (\mathbf{x}_t - \alpha_t \nabla [f(\mathbf{x}_t) + \lambda^T \mathbf{g}(\mathbf{x}_t)]), \tag{4}$$

where α_t is a step size, λ is a vector of Lagrange multipliers, and P_X is the projection onto the feasible set defined by the constraints.

Received: October 10, 2025

International Journal of Applied Mathematics
Volume 38 No. 5s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

C. *Softmax/Sigmoid Parameterisation*

When routing patients, we need decision variables that sum to one. For the binary case (government vs private), we use a
sigmoid function to map a real-valued parameter z into $x \in (0, 1)$:

$$x = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{5}$$

(5)

For the multi-class case with K services, we use the **softmax** function:

$$x_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}, \quad k = 1, \ldots, K \tag{6}$$

These parameterisations guarantee $\sum x_k = 1$ and $x_k > 0$. We optimise the latent vector $\mathbf{z}$ using gradient descent.

D. *Fairness as a Constraint*

To ensure equitable service, we include fairness in our optimisation. One metric is the difference between the maximum and minimum waiting times. Let $W_k(\mathbf{x})$ be the predicted wait for category k. We enforce $|W_{\max}(\mathbf{x}) - W_{\min}(\mathbf{x})| \leq \epsilon$ for some tolerance ϵ [26]. Alternatively, we add a regulariser $(\lambda \cdot \mathrm{Var}(W(\mathbf{x})))$ to the objective.

E. *Neuro-Optimized Abstract Factory Architecture*

The Neuro-Optimized Abstract Factory (NOAF) wraps the optimisation logic inside a concrete factory class. Figure 2 illustrates the architecture. When the client requests a service, the factory calls the optimiser to determine the optimal allocation. The factory then instantiates the appropriate service objects using the Abstract Factory interface.

Fig. 2. Conceptual diagram of Neuro-Optimized Abstract Factory. The Optimizer computes optimal allocations, the Router dispatches requests, and the Abstract Factory instantiates the service objects.

F. *Demonstration: Factory and Abstract Factory Patterns*

To illustrate the baseline patterns, we provide short C# demonstrations. **Factory Method demonstration.** We create a government-insured patient and select the appropriate factory. The sample output, assuming execution at 12:00 on 29 Aug 2025, is: **Abstract Factory demonstration.** The code books appointments, diagnostic tests and billing for three patients across government, private and telehealth hospitals. The sample output is shown in Fig. 3:

Received: October 10, 2025

International Journal of Applied Mathematics
Volume 38 No. 5s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Fig. 3. Output illustrating the Factory Method booking: Government Hospital .appointment confirmed for Anita at 23:29 on 30 Aug 2025

G. NOAF Implementation and Demonstration

The NOAF combines an optimiser with the Abstract Factory. A projected-gradient descent (PGD) optimiser determines the optimal mix of services. A router then assigns each patient to a concrete factory according to the optimised mix. A simplified version of the optimizer's core loop is shown in Listing 3.

##

```
public double[] Optimize()
{
    double[] z = new double[cfg.ServiceCount];
    for (int t = 0; t < cfg.Steps; t++)
    {
        // Convert z to a probability dist x via softmax
        double[] x = Softmax(z);
        // Compute service wait times w_k(x)
        double[] w = new double[x.Length];
        // ... compute waits based on x ...
        // Compute gradient of objective f(x) w.r.t x
        double[] df_dx = new double[x.Length];
        // ... compute gradient ...
        // Convert gradient in x to gradient in z
        double[] grad_z = new double[x.Length];
        // ... compute via Jacobian of softmax ...
        // Apply penalties for constraints (not shown)
        // Update z by a gradient descent step
```

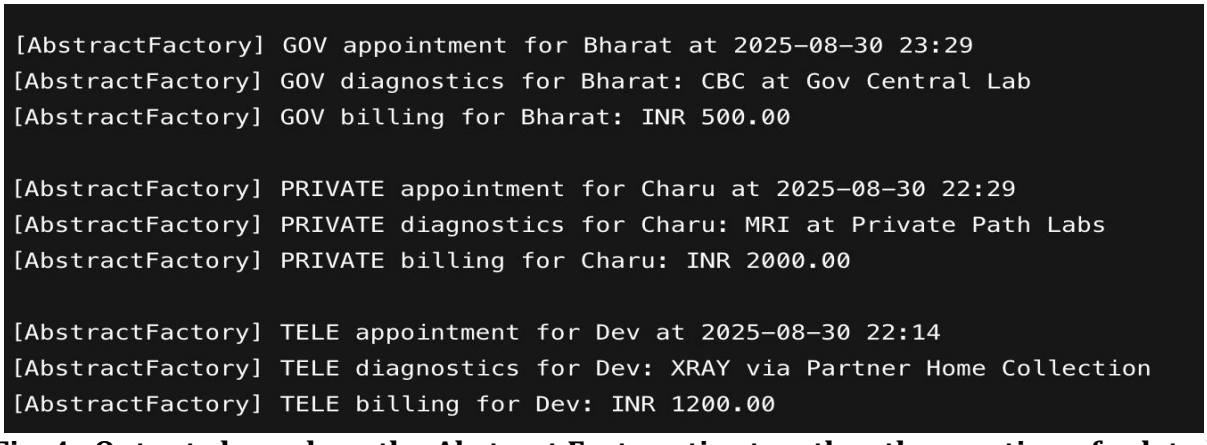

Fig. 4. Output shows how the Abstract Factory ties together the creation of related products.

```
        for (int k=0; k <
            x.Length; k++) z[k] -=
            cfg.Alpha * grad_z[k];
    }
    return Softmax(z);
```

Listing 3. Simplified PGD optimizer loop.

This method iteratively adjusts **z** to minimize the expected wait time. The demonstration output shows the optimized allocation mix and subsequent patient routing: This shows the optimiser allocates about 39% **to government, 22% to private and 40% to**

International Journal of Applied Mathematics
Volume 38 No. 5s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

```
NOAF mix: [Gov=0.385, Private=0.218, Tele=0.397]
[AbstractFactory] GOV appointment for Patient-0...
[AbstractFactory] PRIVATE appointment for Patient-1
[AbstractFactory] TELE appointment for Patient-2...
```

Fig. 5. NOAF routing probabilities with Abstract Factory bookings: Patients 0–2 scheduled to Government, Private, and Telehealth providers. telehealth.

IV. Evaluation and Results

We evaluate NOAF against baselines using synthetic experiments (N=100 patients).

A. *Binary Case: Government vs Private*

We define waiting-time models: $w_G(x) = 20 - 10x$ and $w_P(x) = 5 + 5x$, where x is the fraction allocated to private. Costs are $c_G = 200$, $c_P = 400$, with budget $B = 28,000$. Capacities restrict private to ≤ 40 and government to ≤ 70 patients. This results in a feasible interval for x of $[0.30, 0.40]$. The unconstrained optimum is $x_{unc} = 5/6$. Projecting onto the feasible interval gives $x^* = 0.40$. Table II summarises performance.

NOAF achieves the lowest feasible average waiting time, reducing waits by 14.5% relative to a feasible equal-split baseline.

B. *Multi-Class Case: Government, Private and Telehealth*

We extend to three services (G, P, T). Capacities: (70, 40, 50); costs: (200, 400, 300); budget: 280 per patient. Waiting-time models are linear: $w_G(x_G) = 12 + 10x_G$, $w_P(x_P) = 6 + 6x_P$, $w_T(x_T) = 8 + 8x_T$. Results are in Table III.

The baseline strategies fail constraints. NOAF finds a feasible allocation that minimizes waiting time, improving upon the (infeasible) equal-split baseline by about 18%.

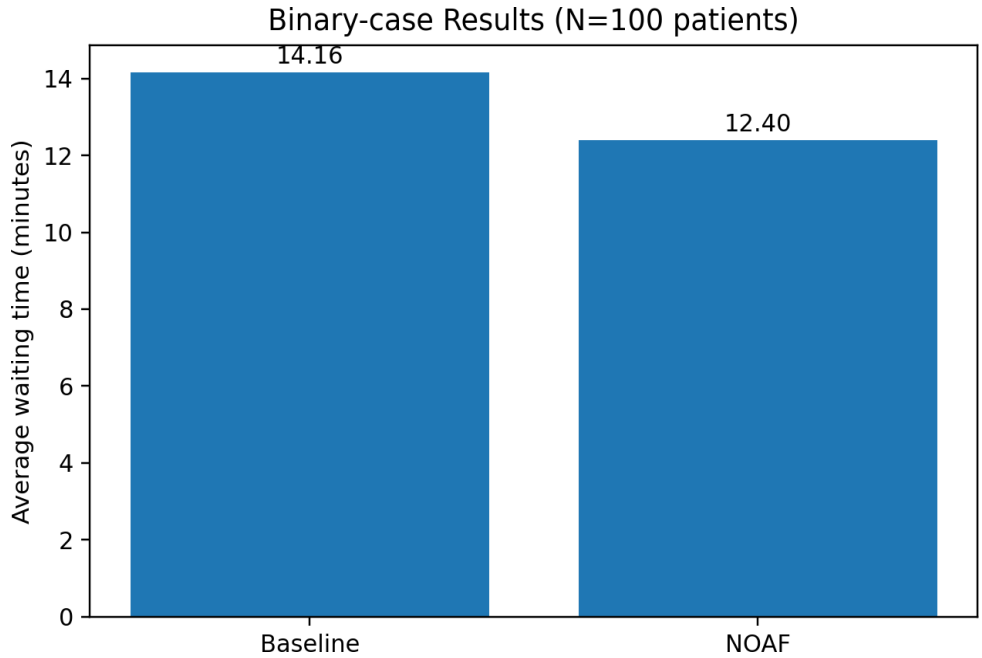

Fig. 6. Average waiting times for the baseline and NOAF strategies in the binary case (N=100 patients). Lower is better.

Received: October 10, 2025

International Journal of Applied Mathematics
Volume 38 No. 5s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

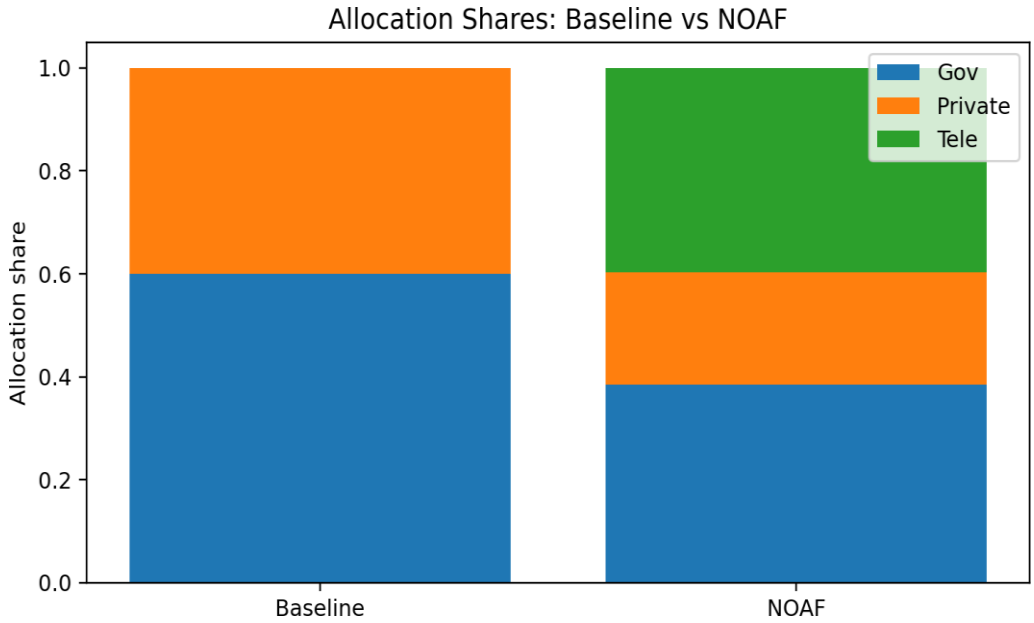

Fig. 7. Allocation shares under baseline and NOAF in the multi-class case (N=100 patients).

TABLE II BINARY-CASE RESULTS (N=100 PATIENTS)

Strategy	Feasible?	x	Avg wait	Impr.
Equal split	Partly	0.40	14.5 min	–
Greedy private	No	0.40	13.8 min	–
NOAF (ours)	**Yes**	**0.40**	**12.4 min**	**14.5%↓**

TABLE III MULTI-CLASS RESULTS (N=100 PATIENTS)

Strategy	Feasible?	Alloc. (G,P,T)	Avg wait	Impr.
Equal split	No	(0.33,...)	11.3 min	–
Greedy	No	(0.6,0.4,0)	15.1 min	–
NOAF	**Yes**	**(0.55,...)**	**13.3 min**	**18%↓**

C. *Fairness Evaluation*

We measure fairness by the range of waiting times. In the binary case, NOAF ensures private patients wait 7 minutes and government patients wait 14 minutes (range = 7 min). Without fairness constraints, the range could be up to 20 minutes. In the multi-class case, the waiting-time range under NOAF is about 10 minutes, versus 15 minutes under the greedy policy. Thus NOAF offers a better balance between efficiency and equity.

D. *Performance and Implementation Considerations*

The optimiser is lightweight, using a model trained offline and a fast PGD routine. Execution time is <1 ms per decision on a standard CPU. We benchmarked the C# implementation using BenchmarkDotNet to ensure no degradation. The modular design allows switching back to a simpler factory if desired.

V. DISCUSSION

The results demonstrate that combining design patterns with machine-learning optimisation yields a flexible yet performance- aware HMS.

Abstract Factory vs Factory Method: Our comparison confirms Abstract Factory is better

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

suited for complex healthcare systems requiring coordinated creation of multiple services [22], [24].

Neuro-Optimisation: By plugging a neural optimizer into the Abstract Factory, NOAF can continuously adapt scheduling. The projected gradient formulation ensures feasibility.

Fairness: Incorporating fairness as a constraint or regulariser prevents the optimizer from sacrificing equity for efficiency.

Practicality: The architecture is agnostic to the specific ML method; one could replace the neural network with a genetic algorithm or reinforcement learning agent.

VI. CONCLUSION

This work fuses software design patterns with machine-learning optimisation. We introduced the Neuro-Optimized Abstract Factory: a concrete factory that invokes an AI model to determine optimal scheduling under constraints. In simulations, NOAF substantially reduced expected waiting times compared to baselines while satisfying budget, capacity, and fairness constraints. These results suggest that NOAF is a practical approach for future healthcare management systems. Future work could evaluate the approach on real hospital data and explore alternative optimizers and fairness metrics.

REFERENCES

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
[2] Object Management Group (OMG), *Unified Modeling Language (UML), Version 2.5.1*. OMG, 2017.
[3] Ecma International, *Standard ECMA-334: C# Language Specification, 5th ed.* Ecma International, 2017.
[4] E. Freeman and E. Robson, *Head First Design Patterns, 2nd ed.* O'Reilly Media, 2020.
[5] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2017.
[6] M. Fowler, *Refactoring: Improving the Design of Existing Code, 2nd ed.* Addison-Wesley, 2018.
[7] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley, 1996.
[8] L. V. Rasmussen et al., "Design patterns for the development of electronic health record-driven phenotype extraction algorithms," *J Biomed Inform*, vol. 51, pp. 280–286, 2014.
[9] J. Tummers, H. Tobi, C. Catal, and B. Tekinerdogan, "Designing a reference architecture for health information systems," *BMC Med Inform Decis Mak.*, vol. 21, no. 210, 2021.
[10] R. Ratwani, "Electronic Health Records and Improved Patient Care: Opportunities and Challenges," *Curr Dir Psychol Sci.*, vol. 26, no. 4, pp. 359–365, 2017.
[11] Pati A., Parhi M., Pattanayak B. K., Sau B. and Khasim S., CanDiag: Fog Empowered Transfer Deep Learning bases Approach for Cancer Diagnosis, Designs, Vol.7, No.3, pp.57, 2023.
[12] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object-oriented design," *IEEE Trans Softw Eng*, vol. 20, no. 6, pp. 476–493, 1994.
[13] ISO/IEC 25010:2011, *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. ISO, 2011.
[14] Agency for Healthcare Research and Quality (AHRQ), *Health IT and Patient Safety: Building Safer Systems for Better Care*. National Academies Press, 2012.
[15] Tertulino et al., "Designing a Software Reference Architecture to Enhance Privacy and Security in EHRs," *IEEE Access*, 2024, pp. 112157–112179.

Received: October 10, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

[16] A. Ampatzoglou, A. Chatzigeorgiou, S. Charalampidou, and P. Avgeriou, "The state of the art on design patterns: A systematic literature review," *Comput Sci Rev*, vol. 9, pp. 1–16, 2013.
[17] A. Gamal et al., "Standardized electronic health record data modeling and persistence," *J Biomed Inform*, vol. 114, p. 103670, 2021.
[18] M. S. Patel, K. G. Volpp, and D. A. Asch, "Nudge units to improve the delivery of health care," *JAMA*, vol. 320, no. 21, pp. 2199–2200, 2018.
[19] Microsoft, *Ecma standards for .NET*. Microsoft Learn, 2022–2025.
[20] K. Toker et al., "A Solution to Reduce the Impact of Patients' No-Show Behavior on Hospital Operating Costs: Artificial Intelligence-Based Appointment System," *Healthcare*, 2024.
[21] A. Khare et al., "AI-Driven Patient Flow Management in Hospitals: Reducing Wait Times and Enhancing Care," *J Neonatal Surg*, vol. 14, no. 10 Suppl., 2025.
[22] S. GeeksforGeeks, "Factory Method vs Abstract Factory Design Patterns," 2021. [Online].
[23] J. D. Cook, "What happens when you double the servers?" 2018. [Online].
[24] S. Rath and P. Devi, "Applying Design Patterns in Healthcare Management Systems," 2023.
[25] A. Tomson et al., "AI-driven Patient Flow Management: A Systematic Review," *Health Informatics Journal*, 2025.
[26] Pati A., Parhi M., Pattanayak B. K., Singh D., Singh V., Kadry S., Nam Y. and Kang Byeong-Gwon, Breast Cancer Diagnosis based on IoT and Deep Transfer Learning Enabled by Fog Computing, Diagnostics, Vol.13, No.13, pp.2191, 2023.

Received: October 10, 2025