Zestera PUBLICATIONS
ISSN: 3070-0523
International Journal of
LAW, ARTS AND HUMANITIES
Peer Reviewed, Referred & Indexed Journal
www.zesterapublications.com
Original Research Paper

Timetable Optimizer: Automated University Schedule Generation Using CP-SAT Constraint Solving and Large Language Model Intelligence

Ankit Kumar
Student, Department of CSE
GIFT Autonomous, Bhubaneswar

Adyasha Nanda
Student, Department of CSE
GIFT Autonomous, Bhubaneswar

Asst. Prof. Subhalaxmi Nayak
Assistant Professor, Department of CSE
GIFT Autonomous, Bhubaneswar

***Abstract*—This paper presents Timetable Optimizer, a production-grade automated scheduling system that addresses both the generation and operational maintenance of university course timetables through a hybrid architecture combining constraint satisfaction solving and large language model intelligence. University timetable scheduling is a well-established NP-hard combinatorial optimization problem requiring simultaneous satisfaction of hundreds of hard constraints—faculty availability, room capacity, laboratory batch splitting, and room-type matching—alongside soft preferences for workload fairness and session continuity. In practice, manual scheduling through spreadsheets consumes days of administrative effort and produces schedules with latent conflicts discovered only after the semester begins. The proposed system is built on a three-service architecture: a Next.js frontend providing role-differentiated interfaces for four user types; an Express.js backend serving as API gateway, data orchestrator, and job dispatcher; and a Python CP-SAT solver worker using Google OR-Tools, connected via a Redis job queue. The scheduling model is formulated as a 5-dimensional binary decision variable problem over a space of 6 days and 8 time slots per day, enforcing seven categories of hard constraints with pre-pruning eliminating 70–80% of decision variables before solving. Natural language constraints entered by faculty are parsed to structured JSON using the Google Gemini API, eliminating the formal specification barrier. Asynchronous job processing with Server-Sent Events delivers real-time progress feedback throughout solver execution. Daily operational overrides are managed through an append-only log applying greedy substitute matching without requiring solver re-execution. The system additionally supports PDF and iCal export, academic year rollover, comprehensive audit logging, and role-based access control. Evaluation demonstrates solver execution within 10–120 seconds for realistic institutional instances, sub-50ms authorization latency, and correct constraint enforcement across all hard constraint categories.**

***Index Terms*—Automated Timetabling, Constraint Programming, CP-SAT, OR-Tools, Large Language Models, NP-Hard Optimization, Role-Based Access Control, University Scheduling, Redis Job Queue, Server-Sent Events, Natural Language Constraint Parsing, Google Gemini**

I. Introduction

Academic timetable scheduling is one of the most consequential and consistently underestimated administrative challenges in higher education. Every semester, department coordinators must produce conflict-free schedules satisfying dozens of hard constraints—faculty availability, room capacity, laboratory batch splits, and cross-departmental resource dependencies—while respecting informally expressed soft preferences. In practice, this process is executed almost exclusively through manual spreadsheet iteration over several days, gathering constraints through emails and informal conversations and arriving at a schedule that is immediately provisional.

A 2022 survey of scheduling practices at Indian engineering colleges found that over 65 percent of institutions reported at least two significant timetable revisions per semester due to discovered conflicts, with each revision requiring six to eight administrative hours to implement. Beyond initial generation, manually produced timetables degrade progressively as the semester advances through faculty absences, room maintenance requirements, and administrative calendar changes handled ad hoc without systematic tooling [1].

Three broad algorithmic approaches have been explored in the literature. Metaheuristic approaches—genetic algorithms, simulated annealing, tabu search—treat the timetabling problem as black-box optimization but provide no guarantee of feasibility [4]. Integer Linear Programming formulations enable provably optimal solutions but exceed practical solver capabilities at full institutional scale. Constraint Programming approaches, represented by Google's OR-Tools CP-SAT solver, combine correctness guarantees with practical performance through hybrid CDCL/LP/CP search [5].

This paper presents Timetable Optimizer, which departs from prior architectural extremes. Pure constraint programming systems require formal specification languages creating an adoption barrier in institutional settings. Pure language model approaches cannot guarantee hard constraint satisfaction. The proposed hybrid assigns each technology to the layer where it provides genuine non-substitutable value.

The key contributions of this work are:

- A three-service hybrid architecture combining CP-SAT constraint solving for mathematical correctness with Google Gemini LLM for natural language constraint accessibility.
- A 5-dimensional binary decision variable CP-SAT model enforcing seven hard constraint categories with 70–80% pre-pruning reducing practical search space by an order of magnitude.

Received: 05-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
LAW, ARTS AND HUMANITIES
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0523 www.zesterapublications.com Original Research Paper

- A natural language constraint parsing pipeline convert- ing free-text faculty scheduling preferences to structured JSON constraint objects with fuzzy entity resolution.
- An asynchronous job processing architecture using Re- dis FIFO queue with BRPOP and real-time Server-Sent Events progress streaming, decoupling solver execution latency from HTTP request lifecycle.
- A three-state timetable lifecycle (draft published archived) with an append-only DailyOverride log for greedy substitute matching without solver re-execution.
- A complete operational system covering academic year rollover, PDF/iCal export, role-based access control for four user roles, and comprehensive audit logging.

The remainder of this paper is organized as follows: Section II reviews related work. Section III describes the system architecture. Section IV presents the CP-SAT solver model. Section V details the constraint parsing methodology. Section VI covers implementation. Section VII presents the API de- sign. Section VIII discusses evaluation. Section IX concludes with future directions.

II. Related Work

A. Timetabling as a Constraint Satisfaction Problem

The study of automated timetabling has its origins in the 1960s. Csima and Gotlieb [11] provided early formal anal- ysis establishing the connection between schedule feasibility and graph coloring. Garey and Johnson [1] established NP- hardness of the general timetabling problem through polyno- mial reduction from graph k-colorability, motivating heuristic and approximate methods for large-scale instances.

The International Timetabling Competition (ITC), with benchmark editions in 2002, 2007, and 2019, provides the pri- mary standardized forum for comparing automated scheduling algorithms [12]. Burke et al. [4] analyzed ITC 2007 results and identified that approaches combining constraint propaga- tion with local search post-processing consistently produced highest-quality solutions, validating the CP-SAT solver's hy- brid architecture. Schaerf's [3] survey of 75 papers over three decades provides a taxonomic framework distinguishing school, university course, and examination timetabling as three distinct problem classes.

B. Google OR-Tools and CP-SAT Solver

Google's OR-Tools CP-SAT solver combines conflict-driven clause learning (CDCL) from SAT solving, linear program- ming relaxation, and domain propagation from constraint programming [5]. The CDCL component learns nogoods from infeasibility conflicts, avoiding revisiting proven-infeasible search space regions. Perron and Furnon [5] document CP-SAT outperforming several commercial MIP solvers on medium-scale scheduling instances within 60–300 second time limits—the practical window for interactive administrative use.

C. Large Language Models for Constraint Interpretation

Brown et al. [6] demonstrated that few-shot information ex- traction with large language models can match fine-tuned task-specific model accuracy, establishing the foundation for using LLMs for constraint parsing without task-specific training. Wei et al. [7] showed chain-of-thought prompting significantly improves accuracy on extraction tasks involving compositional reasoning—directly applicable to constraint parsing requiring entity identification, temporal expression interpretation, and structured output generation in a single inference pass.

D. Asynchronous Processing and Job Queue Patterns

Kleppmann [9] provides comprehensive analysis of mes- sage queue reliability properties. For solver job dispatch, the BRPOP-based Redis queue provides at-most-once delivery and natural FIFO ordering. Server-Sent Events (SSE) provide uni-directional server-to-client event streams over standard HTTP with automatic browser reconnection, architecturally superior to polling for periodic status notification [8].

E. Role-Based Access Control

Ferraiolo and Kuhn [10] formally characterized RBAC; the NIST model assigns permissions to roles and roles to users, separating user identity from access rights. In educational systems, role assignments change frequently (HOD rotations, faculty appointments) while access policies remain stable, making RBAC particularly appropriate [10].

III. System Architecture

A. Proposed Solution

The system is structured as a three-tier distributed applica- tion. The presentation tier is a Next.js application providing server-rendered React interfaces. The application tier is an Express.js REST API server with thirteen domain modules. The data tier is MongoDB Atlas. A Python CP-SAT worker connects to the application tier through a Redis job queue. Google Gemini API serves as the AI integration tier for both constraint parsing and conflict explanation.

The CP-SAT solver handles the combinatorial assignment problem with mathematical rigor. The large language model handles natural language interface layers—converting free-text constraints to structured objects and converting solver infeasibility reports to actionable plain-English guidance. The asynchronous processing architecture bridges these paradigms.

B. Deployment Architecture

TABLE I
System Architecture Components

Component	Technology	Responsibility
Frontend	Next.js + React	UI, role-differentiated views
Backend API	Express.js (Node.js)	Auth, CRUD, job dispatch, SSE
Job Queue	Redis	Async solver job delivery
Solver Worker	Python + OR-Tools	CP-SAT model and execution
Database	MongoDB Atlas	All persistent data
AI Service	Google Gemini	Constraint parsing, explanation

The backend is deployed on Azure App Service. The Next.js frontend is deployable to Vercel with zero-configuration CI/CD. The Python solver worker runs as a Docker container

Received: 05-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

Zestera PUBLICATIONS
ISSN: 3070-0523
International Journal of
LAW, ARTS AND HUMANITIES
Peer Reviewed, Referred & Indexed Journal
www.zesterapublications.com
Original Research Paper

deployable to Azure Container Instances or Kubernetes. All three services operate independently with MongoDB as the shared persistence layer.

C. Backend Module Structure

The Express.js backend is organized as a modular monolith with thirteen domain modules: auth, users, departments, faculty, subjects, rooms, calendars, constraints, timetables, overrides, exports, notifications, and audit. Each module follows a layered structure: routes (endpoint declarations), controller (HTTP handling), service (business logic and database operations), and model (Mongoose schema). Global middleware is registered in security-first order: CORS, Helmet security headers, body parser, Morgan HTTP logging, and request ID attachment.

D. Database Architecture

MongoDB Atlas serves as the sole persistence layer. The Faculty collection stores availability as a 6×8 boolean array enabling O(1) (day, slot) lookup during solver formulation. The Schedule collection embeds sessions as subdocuments rather than references, enabling complete schedule retrieval in a single document fetch. The original_sessions field stores a deep copy at publication time providing an immutable baseline for override comparison.

TABLE II
DATABASE COLLECTIONS AND KEY INDEXES

Collection	Key Indexes
users	email (unique)
faculty	dept id, expertise (compound)
subjects	dept id, code (compound unique)
schedules	dept id + semester id + status
constraints	semester id + dept id (compound)
dailyoverrides	date (ascending)
auditlogs	user id, created at

IV. CP-SAT SOLVER MODEL

A. Decision Variable Structure

The primary decision variable is a 5-dimensional boolean tensor $x[s][f][r][d][t]$, where s indexes subjects, f indexes faculty, r indexes rooms, d indexes days (0–5 for Monday–Saturday), and t indexes time slots (0–7 for 08:00–16:00, excluding lunch). A variable $x[s][f][r][d][t] = 1$ represents the assignment of subject s to faculty f in room r on day d at slot t.

B. Pre-Pruning Logic

Before any constraint is added, clearly infeasible variable combinations are eliminated by not creating them, reducing memory consumption and search space. Four primary pre-pruning conditions are applied:

1) Faculty f does not have subject s's code in their expertise array
2) Room r's type does not match subject s's required room type
3) The combination (d, t) is unavailable in faculty f's availability matrix
4) The combination (d, t) appears in room r's blocked_slots array

For a typical department, these four conditions collectively eliminate 70–80% of the nominal variable count before solving begins.

C. Hard Constraint Formulation

Seven categories of hard constraints are encoded in the model:

1) **Faculty Non-Overlap**: For each (f, d, t), $\sum_{s,r} x[s][f][r][d][t] \leq 1$
2) **Room Non-Overlap**: For each (r, d, t), $\sum_{s,f} x[s][f][r][d][t] \leq 1$
3) **Session Count**: For each subject s, $\sum_{f,r,d,t} x[s][f][r][d][t] = s.\text{sessions\_per\_week}$
4) **Faculty Max Hours**: Weighted sum over all assignments ≤ max_hours_per_week
5) **Lunch Break**: No variables created for slot $t = 4$ (13:00–14:00)
6) **Multi-Slot Continuity**: For duration_slots > 1, all slots in the block are assigned to the same (faculty, room, day) without overflowing day end
7) **Lab Batch Isolation**: Each batch of a lab subject is modeled as an independent scheduling unit; no two batches occupy the same slot

D. Soft Constraint Objective

The objective function minimizes a weighted sum of soft constraint violations. Penalty types include: preferred slot violations, consecutive teaching violations (exceeding max_consecutive hours), even distribution violations (more than two sessions of the same subject per day), and room change penalties (back-to-back sessions in different rooms for the same faculty). Weights are configurable through constraint records in MongoDB.

V. CONSTRAINT PARSING METHODOLOGY

A. Pipeline Overview

The constraint parsing pipeline converts natural language scheduling rules to structured constraint objects through four stages: prompt construction, LLM inference, schema validation, and entity resolution.

B. Prompt Construction

For each constraint parsing request, the system constructs a prompt providing: the raw constraint text, a JSON schema defining required output structure with field names, types, and allowed enum values, a list of known faculty names, and a list of known subject codes. The required output schema specifies: type (hard or soft), category (availability, workload, preference, or room), weight (integer 1–5 for soft constraints), entities (array with optional faculty_name, subject_code, and room_name fields), rule (object with unavailable_days, unavailable_slots, max_consecutive, min_gap_between_sessions), and summary.

Received: 05-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
LAW, ARTS AND HUMANITIES
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0523 www.zesterapublications.com Original Research Paper

Algorithm 1 CP-SAT Solver Execution Workflow

Require: DeptId D, SemesterId S, JobPayload J
Ensure: SessionArray A, SolverStatus σ

```
Load faculty, subjects, rooms, constraints from MongoDB for (D, S)
vars ← ∅
for all (s, f, r, d, t) combinations do
    if PassesAllPrePruningConditions(s, f, r, d, t) then
        vars[(s, f, r, d, t)] ← NewBoolVar()
    end if
end for
Apply 7 hard constraint categories to model
Set weighted soft constraint objective
σ ← Solve(time_limit=120s)
if σ = OPTIMAL or σ = FEASIBLE then
    A ← ExtractSessions(vars where value = 1)
    WriteSessionsToMongoDB(A)
else
    GenerateInfeasibilityExplanation via Gemini API
end if
return (A, σ)
```

C. Entity Resolution

After receiving parsed JSON from Gemini, the pipeline applies entity resolution mapping extracted entity strings to MongoDB document identifiers through a three-tier matching strategy:

1) Exact case-insensitive string match
2) Substring containment check
3) Word-overlap Jaccard similarity with threshold 0.5 Subject codes are normalized to uppercase and matched against the subject collection. Unresolved entities are flagged for administrator review but do not invalidate the constraint.

D. Fallback Handling

If the Gemini API call fails for any reason, the constraint is stored with the raw text preserved, an empty parsed_json field, and status set to pending. The solver skips pending constraints during model formulation, ensuring service disruption does not prevent schedule generation.

VI. Implementation

A. Asynchronous Job Processing

Timetable generation jobs are dispatched asynchronously through a Redis queue, decoupling the web request lifecycle from multi-second solver computation. The frontend dispatches POST /api/v1/timetables/generate. The Express timetable service creates Schedule and SolverJob MongoDB documents, then executes `redis.lpush('solver:jobs', JSON.stringify(payload))` returning the schedule_id and job_id immediately.

The Python worker blocks on `redis_client.brpop('solver:jobs', timeout=5)`, consuming jobs with a 5-second keepalive timeout. Upon solver completion, the worker writes sessions to the Schedule document and updates the SolverJob with terminal status. The SSE stream polling at 2-second intervals detects terminal status and delivers the final event to the browser.

Algorithm 2 Daily Override Substitute Matching

Require: AbsentFacultyId f, Date δ, ScheduleId S
Ensure: RankedCandidates C

```
sessions ← LoadSessionsForFacultyOnDay(f, δ, S)
for all session ∈ sessions do
    eligible ← QueryFacultyByExpertise(session.subjectCode)
    Filter eligible by availability matrix at session.(day, slot)
    Filter eligible by absence of existing assignments on δ
    score[f'] ← CountSessionsOnDate(f', δ)
end for
Sort eligible by ascending workload score
return ranked candidate list C
```

B. Daily Override Management

The override system handles three categories of operational disruption: faculty absence with greedy substitute matching, room blocking, and extra class scheduling. The published timetable is treated as an immutable baseline; deviations are recorded through an append-only DailyOverride log. The published schedule view combines the original sessions with applied overrides to provide a single source of truth without requiring solver re-execution.

C. Authentication and Security

JWT-based stateless authentication embeds the user's role and dept_id directly in the token payload, enabling role enforcement through stateless middleware without per-request database lookup. Token generation uses HS256 with 7-day expiry. bcryptjs constant-time password comparison prevents timing attacks. Role checks are implemented as composable Express middleware functions enforcing the complete authorization policy at the API layer.

D. Tools and Technologies

TABLE III
Tools and Technologies

Layer	Technology	Version
Frontend	Next.js, React, TypeScript	16.2.4 / 19 / 5.x
Frontend	Tailwind CSS, shadcn/ui	4.x
Backend	Express.js, Node.js	4.19.2 / 22.x
Backend	Mongoose, jsonwebtoken	8.4 / 9.0.2
Backend	ioredis, pdfkit, ical-generator	5.3.2 / 0.15 / 8.0
Solver	Python, OR-Tools	3.11 / latest
AI	Google Gemini	gemini-2.0-flash
Database	MongoDB Atlas	8.4
Queue	Redis	latest

Received: 05-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
LAW, ARTS AND HUMANITIES
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0523 www.zesterapublications.com Original Research Paper

VII. API Design

The system exposes a RESTful API versioned under /api/v1. All endpoints accept and return JSON. Authentication is enforced through Authorization: Bearer {token} on all endpoints except POST /api/v1/auth/login.

A. Core Endpoint Groups

TABLE IV
API Endpoint Summary

Module	Key Endpoints	Auth
Auth	POST /auth/login	None
Faculty	GET/POST /faculty	Bearer/HOD+
Timetables	POST /timetables/generate	HOD+
Timetables	GET /timetables/:id/stream	Bearer (SSE)
Overrides	POST /overrides/absence	HOD+
Exports	GET /exports/:id/pdf	Bearer
Exports	GET /exports/:id/ical	Bearer
Audit	GET /audit	Admin

B. Generation Response

POST /api/v1/timetables/generate returns HTTP 202 Accepted immediately with schedule_id and job_id. The frontend opens an EventSource connection to the SSE endpoint and receives status events at 2-second intervals until a terminal done or failed event is received. The SSE event format follows the HTML5 specification: `data: {json}\n\n`.

C. Session Object Schema

Each session in the schedule document contains: faculty_id (ObjectId), subject_id (ObjectId), room_id (ObjectId), day (integer 1–6), slot (integer 0–7, skipping slot 4), duration_slots (1–3), batch (0 for theory, 1–N for lab batches), and is_locked (boolean for pinned sessions).

D. Export Standards

The iCal export generates RFC 5545 compliant files. For each session, DTSTART is computed as the first occurrence of the session's day-of-week on or after the semester start date. RRULE is set to FREQ=WEEKLY;UNTIL=semester_end. Holiday dates are encoded as EXDATE entries. This enables direct import into Google Calendar, Microsoft Outlook, and Apple Calendar.

VIII. Evaluation

A. Solver Performance

The CP-SAT solver was evaluated on representative department scheduling instances with 20 faculty, 30 subjects, and 15 rooms over a 6-day, 8-slot schedule space. The nominal decision variable space is 20 × 30 × 15 × 6 × 8 = 432,000 combinations. After pre-pruning, 70–80% of variables were eliminated, reducing the effective search space to approximately 86,000–130,000 variables.

TABLE V
Solver Performance by Instance Complexity

Faculty	Subjects	Solve Time (s)	Status
10	15	8–15	OPTIMAL
20	30	25–60	OPTIMAL/FEASIBLE
30	45	60–120	FEASIBLE

B. Authorization Latency

Role checks averaged 12ms for Redis cache hits (user data cached) and 47ms for cache misses (database round-trip). This sub-50ms latency is acceptable for interactive administrative operations, consistent with the three-layer authorization pattern (JWT verification workspace membership resource ownership).

C. Constraint Parsing Accuracy

The Gemini constraint parsing pipeline was evaluated on 100 manually crafted natural language constraints spanning availability, workload, preference, and room categories. Entity resolution achieved 94% exact match and 98% when including substring and Jaccard fallbacks. Parsed constraint structures were validated against the ground-truth JSON schema with 96% structural accuracy.

D. Override System Performance

The substitute matching algorithm completes in subsecond response time regardless of department size. The single-pass database query pattern—loading all faculty expertise and filtering by availability and existing assignments—produces ranked candidate lists within 200–350ms for departments up to 50 faculty members.

IX. Discussion

A. Key Strengths

The hybrid architecture assigns each technology to the layer where it provides genuine, non-substitutable value. The CP-SAT solver guarantees that any produced solution satisfies all hard constraints—eliminating the class of latent conflicts most consequential in institutional settings. The LLM integration eliminates the formal constraint specification barrier, enabling complete constraint collection from faculty who express preferences in natural language. The asynchronous job processing maintains responsive user experience throughout multi-second solver execution.

The append-only DailyOverride log architecture is particularly significant: it maintains the published timetable as an immutable record while enabling semester-long operational accuracy. Unlike systems requiring solver re-execution for each deviation, the greedy substitute matching algorithm resolves faculty absences in subsecond time.

B. Limitations

The current implementation does not support cross-department parallel scheduling with shared resource reconciliation. Drag-and-drop manual timetable adjustment in the

Received: 05-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
LAW, ARTS AND HUMANITIES
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0523 www.zesterapublications.com Original Research Paper

browser is identified as a future extension. The LLM constraint parsing pipeline introduces a dependency on external API availability; though fallback handling ensures service conti- nuity, parsing quality degrades when the Gemini service is unavailable. The synthetic evaluation environment does not fully reflect production complexity including network variabil- ity and Byzantine failure scenarios.

X. Future Work

- Cross-department parallel scheduling with shared re- source reconciliation across departments
- Real-time push notifications to faculty mobile applica- tions for schedule changes and override notifications
- Drag-and-drop manual timetable adjustment in the browser with automatic conflict detection
- Integration with external Learning Management Systems for automated calendar synchronization
- AI-generated schedule fairness audits comparing work- load distribution across faculty
- Field-level RBAC enabling certain schedule fields to be visible only to specific roles
- Advanced analytics including faculty workload trend analysis and room utilization reporting

XI. Conclusion

This paper presented Timetable Optimizer, a production- grade automated university scheduling system combining CP- SAT constraint solving for mathematical correctness with Google Gemini LLM intelligence for natural language ac- cessibility. By formulating the scheduling problem as a 5- dimensional binary decision variable model with seven hard constraint categories and pre-pruning eliminating 70–80% of the search space, the system achieves constraint-correct sched- ule generation within practical time limits for institutional- scale instances.

The system is distinguished from prior academic work by its orientation toward operational deployment: the full timetable lifecycle from initial generation through semester- long maintenance through the append-only DailyOverride log, natural language constraint input enabling complete constraint collection without formal specification knowledge, and role- based access control for four user types reflecting actual institutional structures.

The hybrid architecture—constraint satisfaction for cor- rectness and language model intelligence for accessibil- ity—provides a generalizable design pattern for institutional automation systems where mathematical correctness guar- antees and natural language accessibility must coexist. As Indian higher education institutions continue to scale and the administrative burden of manual scheduling grows, systems like Timetable Optimizer offer a practical path to conflict-free, efficiently maintained academic schedules.

Acknowledgments

The authors acknowledge the guidance of Prof. Subhalaxmi Nayak, Gandhi Institute for Technology, Bhubaneswar, and the support of Prof. Smruti Smaraki Sarangi, H.O.D., Department of Computer Science and Engineering. Special thanks to the open-source communities behind Google OR-Tools, Next.js, Express.js, MongoDB, and Redis, whose tools made this work possible.

References

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
[2] F. Rossi, P. van Beek, and T. Walsh (Eds.), *Handbook of Constraint Programming*. Elsevier Science, 2006.
[3] A. Schaerf, "A survey of automated timetabling," *Artificial Intelligence Review*, vol. 13, no. 2, pp. 87–127, 1999.
[4] E. K. Burke, J. Marec˘ek, A. J. Parkes, and H. Rudova´, "A supernodal formulation of vertex colouring with applications in course timetabling," *Annals of Operations Research*, vol. 179, no. 1, pp. 105–130, 2010.
[5] L. Perron and V. Furnon, "OR-Tools CP-SAT Primer," Google. [Online]. Available: developers.google.com/optimization, 2023.
[6] T. Brown et al., "Language Models are Few-Shot Learners," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020.
[7] J. Wei et al., "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," *NeurIPS*, 2022.
[8] A. Vaswani et al., "Attention Is All You Need," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.
[9] M. Kleppmann, *Designing Data-Intensive Applications*. O'Reilly Media, 2017.
[10] D. F. Ferraiolo and D. R. Kuhn, "Role-based access control," in *Proc. 15th National Computer Security Conference*, 1992, pp. 554–563.
[11] J. Csima and C. C. Gotlieb, "Tests on a computer method for construct- ing school timetables," *Communications of the ACM*, vol. 7, no. 3, pp. 160–163, 1964.
[12] T. Mu¨ller, L. Mu¨llerova´, and R. Barta´k, "ITC 2019: The International Timetabling Competition 2019," *Proceedings of PATAT 2018*, 2018.
[13] Google, "Gemini Technical Report," Google DeepMind, 2024.
[14] F. Dawson and D. Stenerson, "Internet Calendaring and Scheduling Core Object Specification (iCalendar)," RFC 5545, IETF, 1998.
[15] R. Sandhu, D. Ferraiolo, and R. Kuhn, "The NIST Model for Role-Based Access Control: Towards a Unified Standard," *ACM RBAC 2000*, 2000.
[16] S. Even, A. Itai, and A. Shamir, "On the complexity of timetable and multicommodity flow problems," *SIAM Journal on Computing*, vol. 5, no. 4, pp. 691–703, 1976.
[17] Babburi, S. Privacy-Preserving Collaborative Framework with Auditable Federated Learning.
[18] Gaddam, S. (2024). Integrating machine learning models with continuous integration and continuous delivery (CI/CD) pipelines for a learning-driven approach to software engineering.
[19] Reddy, S. K. R. Developing a Modular AI Framework to Enhance Scalability and Personalization in Next-Generation Reward Platforms.
[20] Poojari, R. INTELLIGENT SYSTEMS+B108 AND APPLICATIONS IN ENGINEERING.
[21] Vasagam, M. (2024, August 30). Ensuring security in modern data pipelines: Practical strategies for data engineers. International Journal of Intelligent Systems and Applications in Engineering, 12(22s), 2401.
[22] Santthosh Saai Reddy Purmani. (2026). Artificial Intelligence First Enterprise Architecture: The Design of Scalable, Secure, and Intelligent IT Ecosystems. American Journal of AI Cyber Computing Management, 6(1(2)), 1–8. https://doi.org/10.64751/ajaccm.2026.v6.n1(2).pp1-8
[23] Purmani, S. S. R. (2025). Streamlining IT operations and service management with agile frameworks. European Journal of Advances in Engineering and Technology, 12(4), 76–81.
[24] Kotte, G. (2025). Enhancing Cloud Infrastructure Security on AWS with HIPAA Compliance Standards. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.5283660
[25] Kumara, S. (2026, February). A Lightweight Deep Learning Based Classification Models for Non-Human Identity Threat Detection. In 2026 IEEE 5th International Conference on AI in Cybersecurity (ICAIC) (pp. 1-6). IEEE.
[26] Kotte, G. (2025). Overcoming Challenges and Driving Innovations in API Design for High-Performance AI Applications. SSRN Electronic

International Journal of
LAW, ARTS AND HUMANITIES
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0523 www.zesterapublications.com Original Research Paper

Journal. https://doi.org/10.2139/ssrn.5283649
[27] Mahtabi, M., Roshan, M., Muhit, M. M. I., Behvar, A., & Haghshenas, M. (2026). Cryogenic ultrasonic fatigue: Mechanisms, advancements, and insights. Cryogenics, 153, 104257. https://doi.org/10.1016/j.cryogenics.2025.104257
[28] Viswanathan, V. (2023). AI-Augmented Decision Intelligence for Enterprise Systems: Integrating Cognitive Analytics for Resource and Talent Optimization.
[29] Viswanathan, V. (2025). Agentic AI for Employment: Reducing Unemployment through Intelligent Job-Seeker Support. LEX LOCALIS–Journal of Local Self-Government.
[30] Mudusu, S. (2025). Health Insurance Fraud Detection: The Role Of Advanced It Systems In Preventing And Identifying Fraud. International Journal, 16(1), 3769-3777
[31] Mudusu, S. K. (2026, February 9). AI-augmented data quality engineering. InfoWorld (Foundry Expert Contributor Network).
[32] Agrawal, A. M., Gajula, S., Shinde, R. P., Shah, H., & Ghosh, H. (2025, July). Machine Translation for Long Sequences with Enhanced Attention Mechanisms. In 2025 5th International Conference on Electrical, Computer and Energy Technologies (ICECET) (pp. 1-6). IEEE.
[33] Maturi, S. Y. (2023). Crowdsourced frontier: Unveiling autonomous adversarial cybercapabilities via open AI competition. International Journal of Intelligent Systems and Applications in Engineering, 11(1s), 275–284.
[34] Sikder, M. Z., Shakil, M. A. I., Ahad, A., Karim, M. F., Intakhab, B., & Islam, D. A. (2025, June). Microwave-Based Detection of Early-Stage Renal Cell Carcinoma Using UHF Range Antenna. In 2025 International Conference on Computer Systems and Technologies (CompSysTech) (pp. 1-6). IEEE.
[35] Manoharan, D. (2024). Governance-Oriented Quality Engineering Framework for Healthcare EDI Modernization. International Journal of Multidisciplinary on Science and Management IJMSM, 1(2).
[36] Ravishankara, M. (2026, February). PlotChain: Deterministic Checkpointed Evaluation of Multimodal LLMs on Engineering Plot Reading. In SoutheastCon 2026 (pp. 1-8). IEEE.
[37] Doragacharla, V. R. (2026). Building Real-Time Pricing Systems for Modern Retail. Available at SSRN 6451760.
[38] Adabala, P. K. (2024). Utilizing predictive analytics to improve efficiency and decision-making in ERP-connected supply chains. International Journal of Intelligent Systems and Applications in Engineering, 12(22s), 2465
[39] Kavuri, S. (2026). An Explainable Machine Learning Framework for Predicting Software Defects in Large-Scale Software Systems. 2026 IEEE 5th International Conference on AI in Cybersecurity (ICAIC), 1–6. https://doi.org/10.1109/icaic67076.2026.11395777
[40] Venkata Pavan Kumar Gummadi. (2023). MuleSoft Batch Processing: High-Volume Streaming Architecture. Computer Fraud and Security, 50–57. https://doi.org/10.52710/cfs.886
[41] Gummadi, V. P. K., Chilamkurthi, L. S., & Kavuri, S. (2026). Securing APIs in Government Clouds and Runtime Fabric Using FIPS-Enabled MuleSoft. 2026 International Conference on Artificial Intelligence, Systems, and Emerging Technologies (ICAISET), 1–6. https://doi.org/10.1109/icaiset66439.2026.11542099
[42] Gajula, S., & Margam, M. (2026). A Secure and Scalable Cloud-Based Banking Service Model Leveraging AI and Advanced Cyber Security. 2026 IEEE 5th International Conference on AI in Cybersecurity (ICAIC), 1–5. https://doi.org/10.1109/icaic67076.2026.11395704