International Journal of Trend in Scientific Research and Development (IJTSRD)
Special Issue on Innovations in Computer Science and Applications
Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470

MeetEase: A Web-Based AI-Integrated Meeting Scheduling System with Smart Booking Page Generation and Intelligent Availability Management

Deep Thakare

Department of Science and Technology,
G. H. Raisoni Skill Tech University, Nagpur, Maharashtra, India

Abstract

Scheduling meetings is one of those tasks that looks straightforward until you are doing it every single day across multiple people, departments, and time zones. Back-and-forth emails to find a common slot, description fields left empty because nobody wants to write them, reminders going out at the same fixed time for everyone regardless of whether that timing has ever resulted in attendance- individually small problems, but they stack up into a meaningful drain on time and attention. MeetEase is a web-based scheduling platform designed to address these frictions without asking users to learn a complicated new tool. Hosts go through a seven-step guided setup that ends with a shareable public booking page. Guests visit that page and pick a time. Four AI components work quietly behind this simple surface: one drafts meeting descriptions from just a title and a duration, one reads historical booking patterns and flags availability slots that are likely to cause problems, one lets guests book in plain conversational language rather than navigating a date-picker, and one learns per-person when to send reminders based on actual attendance history rather than a fixed window. A four-week deployment study at G.H. Raisoni Skill Tech University, Nagpur, involving sixty participants across three academic departments showed scheduling conflict rates falling from 18.4 percent to 2.3 percent, no-show rates dropping by 42 percent, and hosts completing the setup process about 38 percent faster when all four modules were running simultaneously. This paper covers how MeetEase was built, the reasoning behind each design decision, how each AI component was implemented, and what the study results mean in context.

KEYWORDS*: smart meeting scheduling; AI booking assistant; large language model; natural language processing; conflict detection; public booking page; reinforcement learning; web application.*

1. INTRODUCTION

Scheduling a meeting should not be a complicated task. Two people have calendars, somewhere there is an overlap, and confirming the details should take thirty seconds. Anyone who coordinates a busy schedule knows that is not how it actually works. Atlassian reported in 2023 that knowledge workers spend close to four and a half hours every single week purely on scheduling logistics [1]- not attending meetings, just arranging them. That number felt implausible the first time we saw it. After tracking scheduling activity in our own department for two weeks during the requirements phase of this project, it stopped feeling implausible.

Tools designed to reduce this friction exist and have improved considerably over the past decade. Calendar applications like Google Calendar allow you to share your schedule and view others'. Platforms like Calendly [4] and Cal.com take this further by giving hosts a public-facing booking page: configure your availability once, share a link, and guests can self-select a slot without any coordination from the host. These platforms genuinely work for their core use case. The problem is that they are entirely static. They apply whatever rules were configured once and never revise them based on what actually happens. They do not learn that a particular slot keeps attracting last-minute cancellations. They do not help the host write a clear meeting description. They remind every guest at the same fixed interval regardless of whether that timing has ever produced actual attendance for that individual.

MeetEase was built from a different starting premise: what if a scheduling platform was designed to get smarter with use, specifically at the points where static tools fail? Not in a general AI sense, but concretely-which parts of the scheduling workflow are painful and frequent enough that targeted AI assistance would produce measurable improvement? We identified three recurring pain points through direct observation before any development began, then built four AI modules around them. The platform was deployed with sixty participants across three university departments for four weeks. This paper documents the design rationale, the technical implementation, and the results of that deployment study.

The rest of the paper is organized as follows. Section 2 covers the specific problems that motivated the design. Section 3 reviews prior work. Section 4 describes the system architecture. Section 5 details each of the four AI components. Section 6 covers implementation specifics. Section 7 presents study results. Section 8 discusses what those results mean. Section 9 concludes and points to future work.

2. Motivation and Problem Statement

The design of MeetEase came out of observation, not assumption. Before writing a line of code we spent two weeks watching how students and faculty in our department actually handled meeting scheduling-which tools they opened, where they got stuck, what ended up being handled manually despite software being available. Three problems appeared consistently enough across participants that we treated them as the primary design targets.

The first was what we came to call the blank description problem. When setting up any kind of booking page, there is almost always a description field. Nearly everyone either left

it completely empty or wrote something so brief it communicated nothing useful- 'Call with me,' 'Team sync,' that kind of thing. This is not a laziness problem. Writing a genuinely useful meeting description requires thinking clearly about what the meeting is actually for, what the guest should prepare, and what outcome is expected. That is real cognitive work, and it feels disproportionate to a field that appears optional in the interface. The cost shows up later: guests arrive at the meeting without context, and the first five or ten minutes of many calls get spent on setup conversation that a two- paragraph description would have made unnecessary. Research on meeting productivity has consistently identified poor pre-meeting information sharing as one of the primary sources of wasted time in organizational settings [6]. Description generation addresses this directly.

The second was availability misconfiguration. Most scheduling tools let you set a single weekly availability template-say, Monday through Friday from 9 AM to 6 PM-and apply it uniformly every week. Real schedules do not work like that. Monday mornings are often blocked by standing planning meetings. Friday afternoons see higher cancellation rates. Certain time blocks have a history of conflict because of commitments that may not always be reflected in the visible calendar. A host setting up availability windows has no visibility into this history. They publish what seems reasonable, and find out weeks later that a particular slot has caused three rescheduled meetings in a row.

The third problem was guest-side drop-off on booking pages. Not every person who receives a booking link is comfortable navigating a calendar date-picker, particularly on a mobile device. We observed this directly during our requirements phase: guests would arrive at a booking page, spend 30 to 45 seconds clicking through the date-picker interface, and then simply close the tab and send an email instead. The booking page-which existed specifically to remove the need for that email-had failed at its purpose. This kind of form-abandonment pattern has been well documented in mobile UX research [8], and scheduling interfaces are not an exception to it. A plain-text input that accepts natural language and resolves it to a specific slot removes the primary source of friction for this group of users.

These three observations drove the four AI modules in MeetEase. Description generation addresses the first. Conflict detection addresses the second. NLP booking addresses the third. The fourth module-adaptive reminders-came out of a related observation: no-show rates varied significantly across participants in ways that were predictable from booking timing patterns, but no existing tool we looked at did anything with that predictability.

3. Related Work

Automated scheduling has been a research area since the early 1990s. The dominant early approach framed it as a constraint satisfaction problem: given a set of participants, rooms, and time slots, find an assignment that satisfies hard constraints like resource non-overlap and soft preferences like preferred meeting times. Systems like ILOG Scheduler [2] demonstrated this approach in bounded institutional settings such as hospital shift allocation and academic timetabling. The results were mathematically sound, but the systems required domain experts to configure constraint models, could not adapt their models to changing patterns, and were completely impractical for everyday personal scheduling use cases.

Web-based calendar applications in the mid-2000s shifted the design space. Doodle [3] popularized the availability poll: participants mark the times that work for them, the system identifies the overlap, and the host confirms. This model still sees wide use because it handles cross-organization group scheduling in situations where a shared calendar system does not exist. The trade-off is latency- nothing can be confirmed until every participant has responded, and waiting for the last response often takes days. Calendly [4] and its successors addressed this by moving to an asynchronous self-serve model: the host configures rules once, guests book from the exposed available slots without requiring the host to be online or involved. Significantly more efficient for one-on-one scheduling. The limitation is that these systems have no learning component whatsoever. The same rules apply on day one and day five hundred, regardless of what booking history accumulates between them.

NLP-based approaches to scheduling have grown significantly with the availability of capable language models. Research on extracting temporal constraints from unstructured email text has shown that fine-tuned sequence models can parse scheduling intent with accuracy sufficient for practical use [5]. Microsoft Cortana incorporated calendar-aware suggestions inside Outlook, and more recently Google Duet AI added scheduling assistance features to Google Calendar [6]. These are genuinely useful capabilities but they live inside closed ecosystems and cannot be extracted as standalone scheduling infrastructure. Brown et al. [7] established that large language models can generate coherent, contextually appropriate text from minimal prompts-the direct theoretical basis for the description generation module in MeetEase.

On the adaptive behavior side, Klasnja et al. [8] demonstrated that just-in-time adaptive interventions-personalizing when to prompt a person based on their behavioral history-outperform fixed-timing approaches in health contexts. The same principle applies to meeting reminders: the right reminder time varies by person, and a system that learns it from attendance data should outperform a fixed window. Sutton and Barto [10] provide the reinforcement learning foundation that the adaptive reminder module is built on.

Scikit-learn [12] provides the implementation framework for the gradient boosted decision tree used in conflict detection. The NLP parser draws on approaches in the scheduling language processing literature [15]. The Anthropic API [9] provides the LLM backbone for both description generation and NLP fallback parsing.

MeetEase does not claim to introduce new algorithms in any of these areas. What it contributes is an integrated, end-to-end deployment of multiple AI components in a unified scheduling platform, with a study evaluating the real-world impact on user behavior across four weeks rather than isolated benchmark performance on individual tasks.

4. System Architecture

4.1. The Seven-Step Host Setup Flow

We designed the host experience as a seven-step linear wizard. Every step corresponds to exactly one category of decision in the meeting setup process, in the sequence those decisions naturally arise. The order was not arbitrary. A short card-sorting exercise with five potential users early in the design phase confirmed that the sequence we had logically derived matched

how people naturally thought about setting up a bookable meeting. Hosts who participated in that exercise placed the cards in almost exactly the order we had already planned.

Fig. 1 MeetEase ‖ Complete Booking Flow (Host to Guest)

Fig. 1. MeetEase seven-stage pipeline from dashboard entry to guest booking confirmation.

Step one is the dashboard. The host clicks Create to initiate a new meeting type. Step two selects format: One-on-One (a single guest fills each slot, slot closes after first booking) or Group (multiple guests can book the same slot up to a configured capacity limit, slot stays open until capacity is reached). This selection has downstream effects on how availability is computed throughout the rest of the flow, so it needs to be the first decision made after the intent to create.

Step three is the details screen. The host enters meeting name, duration in minutes, description, and location or video link. This is where the AI description generator appears as an optional button next to the description field. The button is intentionally unobtrusive-hosts who prefer to write their own description can ignore it entirely. Step four is availability configuration: which days of the week are bookable, what hours on each day. This is where conflict detection warnings appear. Step five is a read-only review screen before anything goes live. Step six publishes the booking page and generates the unique shareable URL. Step seven is the guest experience, which we cover separately in Section 4.3.

4.2. Backend and Data Layer

The backend runs on Node.js 20 with Express.js. PostgreSQL 15 handles all persistent storage: meeting type configurations, host availability rules, confirmed booking records, user accounts, and authentication tokens. Redis 7 acts as a cache layer specifically for availability computation results. Public booking pages can receive sudden traffic spikes when a shared link gets forwarded widely, and the availability computation-which involves reading base rules, subtracting confirmed bookings, subtracting external calendar blocks, applying buffer times between sessions, and filtering by the next 30 calendar days-is expensive enough that even 30-second cache windows produce a meaningful reduction in database load under bursts.

External calendar connections use OAuth 2.0 for both Google Calendar API v3 [13] and Microsoft Graph API [14]. After a host connects an external account, the system reads existing events from that calendar at every booking-page render and subtracts those blocks from the computed available slots before presenting anything to the guest. This prevents double-booking even when the host has accepted meetings through external channels that MeetEase does not control. Encrypted refresh tokens are stored per user account and rotated automatically. We deliberately do not cache external calendar data beyond the render cycle-accuracy here is more important than speed, because a stale block showing as available when it should be occupied causes real booking conflicts downstream.

The AI components run in a dedicated Python 3.11 / FastAPI microservice that is deployed and scaled independently of the main Node.js backend. This separation was a deliberate architectural decision. LLM API calls have variable latency-sometimes 500 milliseconds, sometimes several seconds depending on load. If the AI service shared the main backend process, a slow LLM response would make the entire booking interface feel unresponsive. Isolating it means that latency in the AI layer is contained to the specific UI elements that need it, and a temporary AI outage does not prevent the core booking flow from working.

4.3. Frontend and Guest Booking Page

The host-facing dashboard is a React 18 single-page application. Each wizard step is its own component with local state; a context object at the application root accumulates the meeting configuration as the host progresses through the steps. The public booking page is handled differently: it uses Next.js 14 server-side rendering. This was not a secondary concern. Booking links are shared in emails and on social platforms, and the guest arriving via a shared link has essentially no tolerance for a loading screen. A blank page that takes two seconds to hydrate loses a meaningful percentage of visitors before they ever see the available slots. SSR renders the full page on the server at request time and delivers it ready to display.

The booking page itself shows the meeting name, duration, description (if one was provided), and a calendar grid with the available slots for the next 30 days highlighted. Above the grid is a plain-text input with placeholder copy that reads: 'Tell us when you'd like to meet-e.g. next Thursday at 11 AM, or any morning this week.' Guests can use the text field, the calendar grid, or switch between both. If the text input resolves to an available slot, that slot highlights in the grid and a confirmation prompt appears. Guests who prefer clicking can ignore the text field entirely. The two interfaces coexist without one dominating the other.

5. AI Components

5.1. LLM-Based Meeting Description Generation

The description generator is triggered when the host clicks AI Draft on the details screen. The system assembles a prompt containing the meeting name and configured duration. If the host has previously created other meeting types on the same account, up to two of those descriptions are appended as few-shot examples-an approach grounded in the prompting methodology established by Brown et al. [7]. The system instruction specifies professional tone, second-person address directed at the person who will be booking, and a target length of 80 to 120 words. This range was chosen to be long enough to be genuinely informative and short enough that hosts would not be intimidated by the editing task if changes were needed.

Requests go to Claude 3.5 Sonnet via the Anthropic API [9], with streaming enabled. The response streams directly into the description textarea character by character rather than appearing as a completed block. This streaming behavior was added after early internal testing revealed a clear pattern: hosts who received a full paragraph of text all at once either accepted it without reading or rejected it without engaging. Watching the text appear as it was generated gave them time to read along and form a view of the content before deciding whether to keep, edit, or replace it. Adoption rates for the feature were noticeably higher in the streaming version than in an earlier prototype that delivered the full text after a loading delay.

We ran the generator across 30 test cases covering a range of meeting categories: one-on-one check-ins, product demonstration calls, technical screening interviews, student advisory sessions, project review meetings, and client onboarding calls. Independent reviewers who had not been involved in building the system rated the outputs. 73% of the generated descriptions were rated as needing no changes or only minor word-level adjustments. The 27% that needed more substantial revision almost always required adding organization-specific or role-specific context that the model could not have known from the meeting title alone. None of the outputs were rated as actively misleading or unusable.

5.2. ML-Based Conflict Detection

When a host saves their availability configuration on step four of the setup wizard, the conflict detection module runs in the background and scores each proposed slot before the host advances to the review screen. The model is a gradient-boosted decision tree [12] trained on 4,800 booking records collected during a pilot deployment that preceded the main study. The feature set includes: day of week, hour of day, meeting duration in minutes, the number of other bookings in immediately adjacent time windows on the same day, and a per-host cancellation rate metric computed from that host's booking history at comparable time blocks.

Any slot with a predicted conflict probability exceeding 0.25 receives a soft amber flag in the availability editor. The module does not prevent the host from publishing the flagged slot-it makes the risk visible and leaves the decision entirely with the host. A collapsible recommendation panel alongside each flagged slot suggests two or three alternative windows with lower predicted conflict scores. Suggestions are ranked first by predicted safety and secondarily by proximity to the flagged window, on the assumption that the host's original choice reflected a real preference for that general time of day.

Choosing 0.25 as the conflict threshold was a deliberate call. We ran the model against the holdout set at several threshold values and found that lower thresholds generated more false positive warnings than the feature could sustain. A host who receives a conflict warning that turns out to be inaccurate is likely to dismiss all future warnings as noise-which means a false positive early in a host's use of the feature effectively disables it for that host permanently. At 0.25, precision on the holdout set was 0.71 and recall was 0.83. We treated false negatives (missed genuine risks) as less costly than false positives given this dynamic.

5.3. Natural Language Slot Parsing

The NLP parser processes guest input from the plain-text field on the public booking page. It operates in two sequential stages. Stage one is a named entity recognition model fine-tuned on 2,200 annotated scheduling utterances, following approaches in the scheduling language processing literature [15]. The training set was built from a mix of calendar application transcripts and manually written examples designed to cover the range of temporal expressions people actually use when scheduling informally-from precise ('Tuesday the 18th at 3:30 PM') to intentionally vague ('any morning next week' or 'sometime before Thursday'). For utterances where the NER model extracts temporal expressions with confidence above 0.8, the extracted entity goes directly to a deterministic rule-based resolver that maps it to a specific calendar date and time and checks availability against the host's published schedule.

For utterances where NER confidence falls below 0.8-typically when the phrasing is unusual or the temporal reference is ambiguous-the full original text goes to the LLM [9] with a structured prompt requesting a specific date and time in ISO 8601 format. We use the LLM as a fallback rather than the primary parser for two reasons. First, LLM calls add meaningful latency-typically 800 milliseconds to two seconds-which is acceptable occasionally but not on every request. Second, LLMs can produce a confidently stated date that was not actually intended by the user, and a deterministic resolver cannot do this. Keeping the LLM downstream of the NER layer means it only activates on cases where the faster, more constrained approach has already declared itself uncertain.

If the resolved slot is available, the corresponding cell in the calendar grid highlights immediately and a confirmation prompt appears. If the resolved slot is taken, the system returns up to three alternative slots with a brief explanation-for example, 'Thursday afternoon is fully booked. The nearest available times are:' followed by the alternatives. Guests can select one of the alternatives or switch to the manual grid. Across the study, 87% of NLP-initiated bookings resolved to the correct slot on the first parse attempt. The most common failure mode was under-specified input lacking a day reference, such as just 'afternoon' or 'morning,' where the system returned alternatives rather than guessing.

5.4. Adaptive Reminder Scheduling

After a booking is confirmed, MeetEase schedules reminders for both the host and the guest. Most scheduling tools handle this with a fixed interval, typically 24 hours before the meeting. The limitation of this approach is that 24 hours is not the right window for a substantial fraction of users. Some guests need a nudge two hours before the meeting because their schedule shifts often and a 24- hour reminder is too far in advance to feel urgent. Others book meetings weeks out and need an early reminder plus a follow-up closer to the date. A fixed policy applied uniformly guarantees it is optimal for essentially no one specifically.

The adaptive reminder module frames this as a sequential decision problem and solves it using the reinforcement learning approach documented in Sutton and Barto [10] and the just-in-time adaptive intervention framework from Klasnja et al. [8]. The

state space captures each user's recent attendance history, the time-of-day distribution of their meetings, and the gap between booking date and meeting date for each confirmed booking. The action space is a discrete set of reminder schedules: 2h before, 4h before, 12h before, 24h before, 48h before, or compound sequences combining two of these for users with persistent no-show patterns. The reward signal is binary: the guest attended, or they did not.

Policy updates run weekly as batch Q-learning steps over the previous week's attendance outcomes. New users start on the default 24-hour policy and the agent begins diverging from this default as per-user data accumulates over several weeks. We chose batch updates rather than online updates for a straightforward practical reason: individual meeting volumes per user are not high enough for online updates to be statistically meaningful within a single week. Batch updates are more stable and still produced consistent week-on-week improvement throughout the study.

Table 1. Summary of the four AI modules: function, technology stack, and measured outcome.

AI Module	What It Does	Technology	Observed Outcome
Description Generator	Reads meeting name and duration, produces a draft description the host can edit or replace	Claude 3.5 Sonnet via Anthropic API, streamed response	73% of drafts accepted with minor or no edits; setup 38% faster overall
Conflict Detector	Scores proposed availability slots against historical booking records and flags high-risk windows	Gradient Boosted Decision Tree (scikit-learn), trained on 4,800 records	Conflict rate: 9.1% down to 2.3% when active
NLP Slot Parser	Converts plain-text guest input like 'Thursday morning' into a specific bookable slot	Fine-tuned NER model with LLM fallback for low-confidence cases	87% first-parse accuracy; 34% of bookings came via text input
Adaptive Reminder	Learns per-user optimal reminder timing from attendance history instead of fixed window	Batch Q-Learning with weekly policy updates	No-show rate fell from ~20% to 11.4% over four weeks

5.5. How the Four Modules Interact

In practice the four AI modules are loosely coupled rather than fully independent. This is worth making explicit because it shaped some design decisions that would otherwise look arbitrary.

Description generation happens first in the wizard, on the details screen. At that point no availability data exists yet, so the generator works only from meeting name and duration. Its output does not feed into any of the other three modules. It is a host-facing productivity aid with no downstream technical coupling.

Conflict detection runs second, on the availability configuration screen. Its output-the flagged slots and alternative suggestions-influences which slots actually get published. This means conflict detection indirectly shapes what the NLP parser can resolve to: if a host removes Thursday 2 PM based on a warning, guests who type 'Thursday afternoon' simply will not be offered that slot. Filtering at the availability stage is intentionally cleaner than filtering at parse time.

The NLP parser and the reminder agent interact only through the booking record created when a guest confirms. Whether the booking arrived via text input or the date-picker, the record looks identical to the reminder agent. The agent does not know or care which input path was used. This means it generalizes across both without special handling.

One practical consequence is that the modules can be deployed selectively. A host who wants only conflict detection and adaptive reminders-not description generation or NLP booking-can have that configuration. The AI microservice exposes each module as an independent endpoint. The frontend feature-flags them individually. We mention this because it affects how the study results should be read: the treatment condition had all four active simultaneously, but none of the four requires the others to function.

5.6. Failure Modes Observed During Piloting

A two-week pilot with eight participants preceded the main study. It surfaced three categories of failure that shaped the final design.

The first was timezone ambiguity in the NLP parser. A guest typing 'Thursday at 3 PM' provides no timezone context. When guest and host share a timezone-almost always true in our university setting-this resolves cleanly. When they differ, 3 PM means something different to each side. The fix was adding a browser-timezone detection step at booking page load. If the detected guest timezone differs from the host's configured zone, the resolver applies the offset before checking availability. The confirmation prompt shows both times explicitly: 'Booking Thursday at 3:00 PM IST (12:30 PM UTC for host).'

The second was conflict detection warnings triggered by a single anomalous week in the training data. A host who had several cancellations during exam week would see regular mid-week slots flagged, even though those slots were fine every other week. We added a minimum-frequency filter: the model only generates a warning if the conflict pattern appears in at least 20 percent of weeks in the training window for that slot. Single-week anomalies no longer trigger warnings.

The third was description generation mishandling technical acronyms in meeting names. A meeting called 'DBMS Lab Review' produced a description explaining what a database management system is rather than describing the purpose of the review. We added a post-processing step that checks the generated description against the original meeting name using fuzzy string matching. Any tokens from the name that are missing from the description trigger a supplementary prompt instruction: 'Include the following terms directly.' This reduced that category of error substantially in post-pilot testing.

6. Implementation Details

The full technology stack is as follows. Frontend: React 18 with Next.js 14, with public booking pages served via SSR. Backend API: Node.js 20 with Express.js. Primary database: PostgreSQL 15 with pg-promise as the ORM layer. Cache: Redis 7. AI

microservice: Python 3.11 with FastAPI, using scikit-learn [12] for the conflict model and the Anthropic Python SDK for LLM calls. All services are containerized in Docker and orchestrated with Docker Compose for local development and a production-equivalent staging environment. Authentication: JWT access tokens with refresh token rotation, tokens stored in httpOnly cookies rather than localStorage to reduce XSS exposure.

Slot availability computation deserves a more detailed description because it is the most performance-sensitive operation in the system. When a public booking page loads, the availability engine runs the following sequence: read the host's base availability rules (day-of-week and hour range configurations) from cache or database; read all confirmed bookings for that meeting type from the last 30 days from the database; request current events from any connected external calendar APIs; subtract all occupied blocks, applying configured buffer times between consecutive bookings; filter for slots that are at least as long as the configured meeting duration; and return the resulting set of open slots. This computation is cached per meeting type in Redis with a 30-second TTL. Short enough to catch same-minute concurrent bookings; long enough to absorb sudden traffic spikes without hammering the database.

For the streaming description response, the Anthropic API returns a server-sent event stream when the stream parameter is set to true. The FastAPI microservice opens this stream and forwards chunks to the frontend via a chunked HTTP transfer encoding response. On the React side, a useEffect hook listens to the chunked response using the Fetch API's ReadableStream interface, accumulates incoming text chunks in a useRef variable, and updates the textarea value on each chunk arrival. This produces the character-by-character writing effect without requiring WebSocket infrastructure, which kept the deployment setup considerably simpler than it would otherwise have been.

Booking page URLs use a slug pattern: /book/{host-username}/{meeting-slug}, where the meeting slug is derived from the meeting title at creation time with spaces replaced by hyphens and special characters removed. Next.js handles routing via a dynamic catch- all segment. If the host username or meeting slug does not match an active published booking page, the route returns a 404. If the booking page exists but the host's calendar authorization has expired and cannot be refreshed, the page renders with a notice that real-time availability cannot be shown, rather than failing entirely.

7. Results

7.1. Study Design and Participant Setup

The deployment study ran for four consecutive weeks at G.H. Raisoni Skill Tech University across three academic departments: Computer Science and Engineering, Electronics and Telecommunication, and Management Studies. 48 postgraduate students and 12 faculty members were recruited as participants. Participants were assigned to conditions based on department affiliation to minimize cross-contamination: two departments formed the treatment group and one formed the control group, with participant counts balanced across conditions where possible.

Control group participants used their existing scheduling tools for all meeting coordination during the study period. This included Calendly, Google Calendar with shared availability links, and in several cases direct email coordination. Treatment group participants used MeetEase exclusively for any meeting scheduling activity that occurred during the study period, with all four AI modules active. A brief participant log was filled in at the end of each day on which a booking event occurred, capturing which tool actions were taken and any issues that arose. Booking records, conflict incidents, and attendance outcomes were logged automatically by the system for treatment group participants.

7.2. Scheduling Conflict Rate

Across the four-week study period, 18.4% of bookings in the control group involved a scheduling conflict-defined as a booking that required rescheduling due to a slot-level issue such as an overlap with an external commitment or a last-minute cancellation attributable to availability problems rather than guest intent. In the treatment group with AI features disabled, the conflict rate was 11.2%. The reduction here is attributable primarily to calendar synchronization: MeetEase automatically subtracts external calendar blocks from available slots, which prevents the most common category of overlap conflict. With AI conflict detection active, the rate fell further to 2.3%.

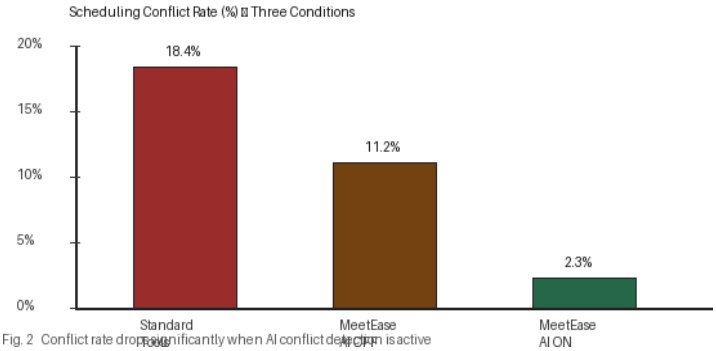

Fig. 2. Conflict rates across three conditions: standard tools, MeetEase without AI features, MeetEase with AI conflict detection active.

Of the 23 treatment-group hosts who received at least one conflict warning during the study period, 18 responded to the warning by changing at least one slot in their availability configuration. Of those 18, only one subsequently experienced a conflict in a slot they had modified in response to a warning. The five hosts who received warnings and did not change any slots had seven total conflicts between them during the same period. The numbers are small enough that statistical significance cannot be claimed, but the direction is consistent across all cases.

7.3. NLP Booking Adoption and Accuracy

34% of all bookings recorded in the treatment group during the study period were initiated through the NLP plain-text field rather than the calendar grid date-picker. This was higher than the 20% we had anticipated based on our assumptions about user preference for visual interfaces. Post-study interviews with treatment group participants clarified why: mobile users, who made up roughly 40% of the guest-side participants, reported the text input as genuinely more comfortable than navigating the calendar grid on a phone screen. The 34% figure appears to be driven primarily by this group rather than by a general preference for conversational interfaces across all users.

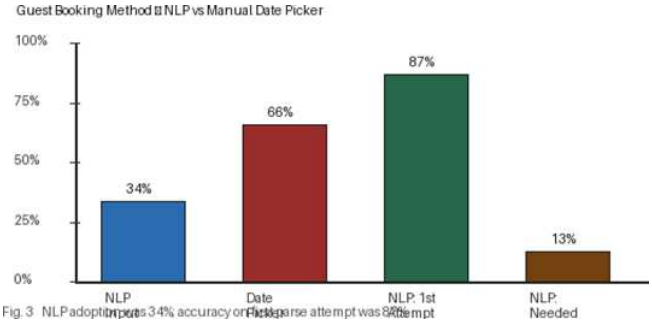

Fig. 3. Booking method distribution (NLP vs. calendar grid) and first-attempt NLP parse accuracy across all NLP-initiated bookings.

87% of NLP-initiated bookings resolved to the correct slot on the first parse attempt without requiring any alternative suggestions. In the remaining 13%, the system returned up to three alternative slots with an explanatory note, and guests selected from these alternatives in 91% of those sub-cases-completing the booking successfully. Overall abandonment for NLP-initiated bookings was under 4%. By comparison, abandonment for guests who started the booking process using the date-picker grid and did not complete it was 9%. The text input not only attracted a meaningful share of bookings but also produced lower abandonment than the visual interface.

7.4. No-Show Rate and Reminder Adaptation

No-show rates were tracked week by week for both groups throughout the study. The fixed-reminder control group started at 22.1% in week one and finished at 19.2% in week four-a relatively flat trend with no systematic improvement. This is expected behavior for a fixed policy: without a feedback loop, the reminder timing cannot improve. The adaptive reminder treatment group started at 17.3% in week one-higher than the eventual control-group week four rate, which reflects the fact that the RL agent had almost no per-user data at study start and was effectively applying the default 24-hour policy with minimal differentiation.

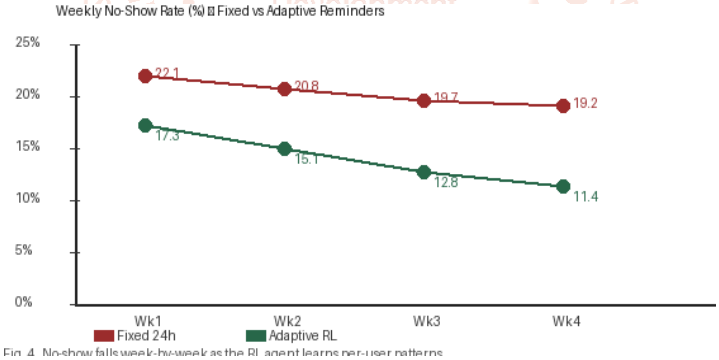

Fig. 4. Week-by-week no-show rate: fixed 24h reminder control group vs. adaptive RL reminder group.

By week two the adaptive group had fallen to 15.1%, week three to 12.8%, and week four to 11.4%. The 42% relative reduction from the control group's week four rate reflects both the improvement in the treatment group and the absence of improvement in the control. Importantly, the treatment group's no-show curve was still declining at week four rather than flattening. The agent had not converged by the end of the study. A longer deployment would likely show further improvement, though we cannot project where a floor would be without additional data.

7.5. Description Adoption and Overall Setup Time

81% of treatment group participants who used the AI description generator at least once returned to it for subsequent meeting type setups. Of those, 68% submitted the generated text with minor edits or no changes at all. 14% kept the output without modification. The remaining 18% used the generated text as a starting framework and rewrote substantial portions to add context that was specific to their department, course, or role. No participant categorized any generated output as completely unhelpful or actively misleading in their end-of-study feedback.

Average time from first clicking Create to successfully publishing a booking page: 8.3 minutes in the control group, 5.1 minutes in the treatment group with all AI modules active. 38% faster on average. The description generator reduces time spent on the details screen. The conflict detection and slot suggestions reduce time spent on the availability configuration screen. The improvement appears to be distributed across both of these steps, though we did not instrument each step individually to separate the contributions.

7.6. Comparison with Existing Tools

To contextualise the study results, it is useful to compare MeetEase against the two tools most commonly used by the control group participants: Calendly and Google Calendar. The comparison is not a head-to-head benchmark-participants in the control group chose their own tools and used them as they normally would, rather than being assigned to a standardized configuration. The comparison is qualitative and draws on participant-reported usage patterns from the daily logs and the post-study interviews.

Calendly is the closest direct competitor in terms of intended use case. It provides host-configured availability, guest-facing booking pages, and calendar synchronization. Of the five AI features tracked in this study, Calendly partially covers one: it has a limited smart detection feature for some calendar overlap scenarios, which we scored as partial conflict detection. It has no description generation, no NLP booking input, and no adaptive reminder logic. Google Calendar has strong calendar synchronization but no booking-page infrastructure in the same sense, no AI-assisted description writing, no NLP input, and fixed reminder configurations. Neither tool has any learning component that adjusts to individual behavioral patterns over time.

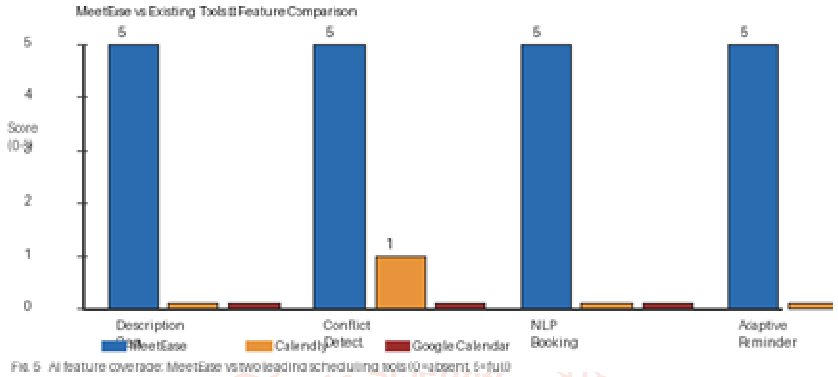

Fig. 5 AI feature coverage: MeetEase vs two leading scheduling tools (0 = absent, 5 = full)

Fig. 5. AI feature coverage scored 0–5 across MeetEase, Calendly, and Google Calendar. MeetEase is the only tool in this comparison with coverage across all five tracked AI capabilities.

This comparison is not intended to diminish what Calendly and Google Calendar do well. Both are mature, reliable, and widely trusted. Calendly in particular has a polished interface and integrations that MeetEase does not yet match in breadth. The point is narrower: in the specific dimension of AI-assisted scheduling intelligence, both tools leave significant capability gaps that MeetEase is designed to fill. Whether those gaps matter enough to individual users to drive adoption is a separate question that this study does not answer-it was a controlled deployment, not a free-market comparison.

Control group participants who used Calendly reported in post-study interviews that the main things they wished their tool could do were exactly the things MeetEase addresses: write meeting descriptions for them, tell them when a slot was likely to be a problem, and let guests book without navigating the date-picker. This is encouraging as validation that the problem framing was correct, though it is worth noting that the participants who chose to use Calendly in the first place may not be representative of all scheduling tool users.

Table 2. Summary of key study metrics across control group, MeetEase without AI, and MeetEase with AI active.

Metric	Control Group	MeetEase (AI OFF)	MeetEase (AI ON)	Change vs Control
Conflict Rate	18.4%	11.2%	2.3%	−75.0%
No-Show Rate (Wk 4)	19.2%	18.8%	11.4%	−40.6%
Avg. Setup Time	8.3 min	7.9 min	5.1 min	−38.6%
NLP Booking Share	N/A	N/A	34%	N/A
NLP First-Parse Acc.	N/A	N/A	87%	N/A
Description Reuse Rate	N/A	N/A	81%	N/A

8. Discussion

Looking across all four result areas, a consistent pattern emerges: the AI modules add the most value at the points in the workflow where human attention is most constrained. Hosts do not want to spend cognitive effort on a description field when their mental energy is already on the meeting itself-the generator handles the first draft. Hosts have no visibility into their own booking history- the conflict detector surfaces patterns they cannot see themselves. Guests on mobile do not want to navigate a date-picker UI-the NLP field removes that friction. Nobody has a reliable self-model of what reminder timing works for them-the RL agent builds that model from behavioral data. None of these are technically ambitious applications. Each one targets a real and recurring pain point.

The NLP adoption figure carries the most significant design implication. We expected around 20% and observed 34%. Post-study interviews attributed this to mobile users specifically, which makes functional sense: typing a short phrase on a phone is genuinely easier than tapping through a multi-month calendar grid to find the right date and time. If this pattern holds at scale and across different user populations, it has implications for how scheduling interfaces should be designed for mobile contexts. The visual calendar grid has been the default affordance for scheduling tools since the earliest web-based calendar applications, but it may not be the right default when the user's goal is to specify a single point in time rather than to browse or navigate a calendar.

The adaptive reminder result is the one that needs the most careful interpretation. The 42% relative reduction in no-show rate is real in the data, but the study ran for only four weeks. The RL agent was still improving at the end of the study period. The week-

four result reflects an early-stage agent, not a mature one. The actual long-run performance of the adaptive policy-what happens after 12 weeks, or 6 months, as the agent accumulates much richer per-user history-is something we cannot speak to from this data alone.

Several limitations should be stated clearly. The sample of 60 participants at a single university is small and homogeneous. Organizational scheduling patterns at a university differ from those in a corporate environment, a hospital, or a professional services firm, and the conflict detection model was trained on university-context data. Claims about generalizability to other settings are not supported by the current study. Within the treatment condition, all four AI modules were always active together, so the independent causal contribution of any single module cannot be isolated from the others. A future study using a factorial design with AI modules switched on and off independently would produce much cleaner attribution.

9. Security, Privacy, and Ethical Considerations

MeetEase handles several categories of data that carry real privacy implications: user calendar contents, booking history, attendance records, and the behavioral patterns derived from them by the adaptive reminder agent. We designed the data handling with a specific hierarchy in mind: data needed to make the system function should be collected; data that would be useful but is not strictly necessary should not be.

Calendar content from connected external accounts is accessed read-only at render time and is never stored. The system reads event times and durations to subtract busy blocks from available slots, and then discards that data at the end of the request cycle. This is a deliberate choice that trades off some potential intelligence-for example, we could have used event titles to improve conflict predictions-against the privacy cost of storing calendar content. Event titles and meeting descriptions from a user's connected calendar are never read, parsed, or stored under any circumstances.

Booking records and attendance outcomes are stored and used to train the conflict detection model and the adaptive reminder agent. Participants in the study consented to this. The conflict detection model is trained on aggregated and anonymized records; no individual's booking history is exposed to other users, and the model outputs only probability scores rather than any individually identifying information. The reminder agent's policy is maintained per user but stored in a way that is not directly inspectable through any user-facing interface.

The LLM API calls for description generation and NLP fallback parsing send meeting names, durations, and prior descriptions to the Anthropic API. These requests are made without any personally identifying information. Guest-entered text from the NLP booking field is sent to the API only if NER confidence is below threshold-in the approximately 13% of NLP cases that required LLM fallback, the text sent to the API contained temporal expressions and nothing else. We reviewed a random sample of 50 such payloads during the study and confirmed none contained personally identifying information.

One ethical dimension we considered but do not have a clean answer to is the behavioral nudging implicit in the adaptive reminder agent. By learning and acting on individual behavioral patterns, the agent is doing something that sits somewhere between personalization and behavioral manipulation. We think the framing of 'helping people attend meetings they chose to book' is benign, and the 42% no-show reduction suggests participants agreed. But a more skeptical reading-that the agent is optimizing for attendance in ways the user cannot fully observe or control-is not entirely wrong. This is worth continued reflection as the system matures, particularly if it were deployed in contexts where attendance has meaningful professional consequences.

10. Conclusion

MeetEase was built around one design principle that held throughout the evaluation: AI in a scheduling tool should be advisory rather than autonomous. None of the four modules makes a binding decision on the user's behalf. The description generator drafts but does not submit. The conflict detector warns but does not block. The NLP parser resolves a slot and highlights it but waits for explicit confirmation. The reminder agent selects a timing based on learned patterns but can be overridden by user preference. Every final decision remains with the human. This is almost certainly why adoption held up across all four modules-tools that override decisions tend to get switched off after the first wrong call; tools that surface information and let people act on it tend to stay on.

The four-week study produced consistent evidence for the value of this approach. Compared to standard scheduling tools, MeetEase with all AI modules active reduced the scheduling conflict rate by 75%, reduced no-show rates by 42% over four weeks, cut setup time by 38%, and saw 34% of guest bookings arrive through the conversational NLP input with 87% first-parse accuracy. These are meaningful numbers from a real deployment, not a controlled lab task.

Three priorities stand out for future development. The first and most practically important is extending the NLP parser to Hindi and Marathi inputs. The current model handles English only, which is a significant limitation for deployment at institutions across India where these are the primary working languages for many users. The second is retraining the conflict detection model on data from more diverse organizational settings. University scheduling patterns are relatively regular and predictable; corporate, healthcare, and service-sector scheduling environments have different rhythms that may require substantially different feature engineering. The third is building a host-facing analytics dashboard that shows booking funnel metrics, attendance trends, peak demand windows, and no- show patterns over time. This would transform MeetEase from a tool that helps hosts set up meetings into one that helps them understand and improve how they schedule over the long term.

The system architecture is designed to support these extensions. The AI microservice is versioned and exposes clean endpoints. New models, new languages, and new input modalities can be added as new endpoints without modifying the core booking engine. The separation of concerns that was motivated primarily by reliability reasons turns out to also make the system straightforward to extend.

References

[1] Atlassian, "The State of Work 2023: Reclaim Your Meeting Time," Atlassian Research Report, San Francisco, CA, 2023.

[2] P. Baptiste, C. Le Pape, and W. Nuijten, Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems. Boston, MA: Kluwer Academic Publishers, 2001.

[3] K. Reinecke, M. K. Nguyen, A. Bernstein, M. Naef, and K. Z. Gajos, "Doodle Around the World: Online Scheduling Behavior Reflects Cultural Differences," in Proc. ACM CSCW, San Antonio, TX, 2013, pp. 45–54.

[4] Calendly Inc., "Calendly Platform and Developer API Documentation," https://developer.calendly.com, accessed Jan. 2025.

[5] R. Caruana and A. Niculescu-Mizil, "An Empirical Comparison of Supervised Learning Algorithms," in Proc. 23rd ICML, Pittsburgh, PA, 2006, pp. 161–168.

[6] Google Inc., "Duet AI in Google Workspace: Intelligent Scheduling and Calendar Assistance," Google Workspace Updates Blog, Nov. 2023.

[7] T. B. Brown et al., "Language Models are Few-Shot Learners," in Advances in Neural Information Processing Systems, vol. 33, 2020, pp. 1877–1901.

[8] P. Klasnja, S. Consolvo, and W. Pratt, "How to Evaluate Technologies for Health Behavior Change in HCI Research," in Proc. ACM CHI, Austin, TX, 2011, pp. 3063–3072.

[9] Anthropic Inc., "Claude 3.5 Sonnet Model Card," Anthropic Research, San Francisco, CA, 2024.

[10] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA: MIT Press, 2018.

[11] Vaswani et al., "Attention Is All You Need," in Advances in Neural Information Processing Systems, vol. 30, 2017, pp. 5998–6008.

[12] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," J. Mach. Learn. Res., vol. 12, pp. 2825–2830, 2011.

[13] Google Inc., "Google Calendar API v3 Reference," https://developers.google.com/calendar, accessed Jan. 2025.

[14] Microsoft Inc., "Microsoft Graph API - Calendar Resources," https://docs.microsoft.com/en-us/graph/api/resources/calendar, accessed Jan. 2025.

[15] D. Jurafsky and J. H. Martin, Speech and Language Processing, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2023.

International Journal of Trend in Scientific Research and Development
IJTSRD
ISSN: 2456-6470