GitHub Link: https://github.com/Josht8601/Adaptive-AI-Scheduler

Project Title: Adaptive AI Scheduler: A Real-Time Personal Assistant for Dynamic Time Management

Project Overview:

The goal of my project is to design and build an AI-powered scheduling assistant that can automatically adapt my schedule in real time. Traditional calendar tools are often too rigid—if I miss a task, fall behind, or something unexpected comes up, my calendar does not adjust, which leads to stress and inefficiency. This project aims to solve that problem by creating a system that distinguishes between fixed commitments, such as classes or meetings, and flexible tasks, such as workouts or study sessions. The system will automatically reorganize my schedule when things change, helping me stay on track and making my day feel more manageable.

By the end of the semester, the system will be able to reschedule at least 80% of missed tasks into available time slots while respecting user-defined constraints. The expected outcome is a prototype dynamic scheduler that is flexible and reliable, ultimately helping users remain productive while reducing the stress of missed activities.

The scope of the project includes creating a calendar view similar to one.uf or Google Calendar that can display both daily and weekly schedules. Events will be categorized into two types: static events, which are fixed in time and cannot be moved, and dynamic events, which are flexible and can be rescheduled when necessary. Users will be able to input preferences and restrictions for dynamic events through a structured form or settings panel. For example, I could require that my daily exercise last 60 minutes and be scheduled between 8:00 AM and 2:00 PM, or I could block off times when no events should be scheduled. The assistant will also prompt me to confirm whether I completed dynamic tasks, and if I did not, it will ask if I want it to reorganize the schedule to fit them elsewhere.

The first version will not integrate with external calendars such as Google Calendar or Outlook. It will not include natural language conversation beyond structured prompts. In addition, the project will not attempt to optimize across multiple users or predict future workloads. For now, the focus will remain on making a single-user, adaptive scheduling assistant that works reliably and efficiently.

The system will use a mix of optimization and AI techniques. Constraint-based scheduling and optimization will form the core of the system, relying on methods such as constraint satisfaction, heuristics, and greedy algorithms to balance competing demands on time. Rule-based logic will be used to distinguish static from dynamic events and to enforce user preferences. While reinforcement learning is not necessary for the initial version, I see it as a potential future extension for personalizing the system’s adjustments over time.

In terms of tools, I will primarily use Python to implement the system. I plan to rely on libraries such as Google's OR-Tools for solving scheduling and optimization problems, as well as Pandas and NumPy for data handling. For the backend, I will consider frameworks like FastAPI to manage inputs and scheduling logic. If I decide to implement a user interface, I will likely use React to provide an interactive calendar view that allows users to visualize and interact with their schedule. To keep the project feasible, I will focus on rule-based logic and OR-Tools first, with reinforcement learning reserved for future development.

Stakeholders:

The project team is the first group of stakeholders, consisting primarily of myself in the role of developer, designer, and tester. My responsibility is to design the system architecture, implement the scheduling logic, and ensure the program functions as intended. My instructor is also an important member of the project team, since they provide feedback, ensure the project aligns with course expectations, and evaluate the final deliverables.

The end users form the second group of stakeholders. I will serve as the initial user to test and refine the system, but the target audience includes students and professionals who manage busy schedules. These users will interact with the assistant by inputting static and dynamic events, setting restrictions and preferences, and responding to prompts that confirm task completion. Their main benefit will come from the system's ability to reorganize schedules on the fly when conflicts, delays, or new priorities arise.

The third group of stakeholders includes other relevant parties that could be involved in the future. For example, data providers could play a role if the system is later integrated with external platforms such as Google Calendar, Outlook, or productivity apps. Regulators might also become stakeholders if the system were expanded to a commercial product, requiring compliance with data privacy and security standards. External partners, such as companies developing productivity tools, could also become stakeholders if they saw potential for integrating an adaptive scheduling assistant into their platforms. In addition, productivity researchers or time-management coaches could serve as validators of the system's usability and effectiveness, strengthening its design and applicability.

Computer Infrastructure Considerations

Project Needs Assessment

The goal of my system is to create an adaptive scheduler that can reorganize tasks in real time when unexpected changes happen. The main tasks include identifying which events are fixed like classes or meetings, which are flexible, and then rescheduling the flexible ones using constraint satisfaction and optimization techniques. The data I'll be working with is mostly structured text (event names, times, and preferences) rather than things like images or audio.

For performance, I want the system to be able to reschedule a missed task in under 500ms, since delays would make it feel unresponsive. Accuracy-wise, my target is that at least 80% of missed tasks get successfully rescheduled into valid slots, which matches the overall project objective. At this stage, everything will run locally on my computer, so I'm not dealing with major deployment constraints like network or GPU access.

I did consider aiming for higher accuracy benchmarks closer to 90–95%, but that would require more advanced optimization and possibly learning-based methods, which aren't feasible within the timeline. Research and benchmarks for Google OR-Tools show that constraint-solving problems of this scale can be done in sub-second latency on CPUs, so that gives me confidence in the target. My priority is responsiveness and reliability over maximum accuracy. If rescheduling ever takes longer than one second consistently, that would be the point where I'd need to rethink the approach.

According to Google OR-Tools benchmarks for constraint-based scheduling, small-scale problems typically achieve sub-200 ms solve times on a 2.5 GHz CPU. Based on this, my goal is to keep average rescheduling latency under 500 ms, with 95th percentile latency under 700 ms. Throughput should sustain at least 20–30 rescheduling operations per second on a single CPU core. These benchmarks confirm that the system's responsiveness and target of rescheduling 80% of missed tasks is realistic for CPU-only execution.

Hardware Requirements Planning

Since this project doesn't require training a large model, I don't need specialized GPUs. Everything can run on a CPU. A basic setup of a quad-core processor (like an Intel i5), 8GB of RAM, and a few hundred megabytes of storage is enough for the prototype. Inference is lightweight because it's just optimization logic and rule-based scheduling.

I considered running it in the cloud with GPU support, or even making an edge/mobile version, but for now those add complexity without much benefit. Benchmarks show that OR-Tools handles this kind of problem just fine on a CPU, so local deployment makes the most sense. The tradeoff here is keeping it simple and cost-effective rather than building something overpowered.

If the project expands to handle thousands of users or tasks, then I'd need to reconsider moving to GPU or cluster-based solutions.

Software Environment Planning

For the development stack, I'm using Python with libraries like OR-Tools, Pandas, and NumPy for optimization and data handling. If I make a backend, I'll use FastAPI, and for a frontend calendar view, I'll use React. I'm developing on Ubuntu since it's lightweight and has strong compatibility with Python libraries, but everything should also run fine on Windows.

I briefly thought about using TensorFlow or PyTorch to add a learning component, but that's beyond the scope of this version. I'll also hold off on containerization tools like Docker for now, though they could be helpful if I want to make the system more portable in the future. The main tradeoff is sticking to a simple Python environment that works well now, while knowing that version mismatches or dependency issues could become a risk later.

Cloud Resources Planning

At this stage, the system runs entirely locally, but if I ever scale it up for multiple users, I will probably use Google Cloud. Since OR-Tools is a Google project, it fits naturally with their ecosystem. Services like Cloud Run or Firestore could handle scaling and storage if I needed them.

I did look at AWS and Azure as alternatives. AWS SageMaker has strong ML features, and Azure is tightly integrated with Microsoft services, but for this project, GCP seems the most straightforward. Costs for a basic VM setup on GCP are also very low so that's another advantage. The tradeoff here is convenience and cost over flexibility across providers. If costs ever went significantly over budget, or if I had to integrate with more Microsoft tools, I might reconsider.

Cost – Performance Comparison

Deployment Type	Avg. Latency	Monthly Cost (Est.)	Scalability	Notes
Local CPU	400 – 700 ms	$0	Limited	Ideal for prototype testing
GCP VM	300 – 600 ms	~$25	Moderate	Enables shared multi-user testing
Cloud Run + Firestore	300 – 500 ms	~$50	High	Supports container scaling and logging

This table quantifies the tradeoffs among latency, cost, and scalability. It demonstrates why local deployment is optimal now but also defines thresholds that would justify cloud migration.

Scalability and Performance Planning

For scaling, the most realistic path would be to containerize the system and use auto-scaling tools like Cloud Run or Kubernetes if I ever move it to the cloud. For now, a single-user setup just

needs efficient optimization, but for larger use cases, I could add heuristics or even reinforcement learning to improve personalization.

I plan to monitor latency, throughput, and rescheduling success rate as the main performance metrics. Initially, I'll log these results locally, but in the future, tools like Prometheus or Grafana could help track them in a cloud environment. I chose to focus on keeping the system fast and reliable, even if that means it won't always find the "perfect" schedule. If latency ever gets worse than 150ms per reschedule request under load, or if the success rate drops significantly, that would be the trigger to rethink the optimization or scaling strategy.

Performance and Risk Triggers

Metric	Threshold	Triggered Action
Average latency > 1 second	Persistent delay in task rescheduling	Optimize OR-Tools solver parameters or add caching for recent constraint sets
CPU utilization > 85% during repeated scheduling tasks	Local resource saturation	Move to containerized deployment on GCP Cloud Run or Kubernetes
Monthly cloud cost > $50	Budget threshold exceeded	Reevaluate hosting strategy and compare GCP vs. AWS/Azure pricing
Rescheduling success rate < 80%	Drop in valid reallocation outcomes	Review constraint weighting and improve conflict-resolution heuristics
Memory usage exceeds 1 GB	Performance degradation during optimization	Profile code for memory leaks or reduce task search space
Latency variance > 300 ms over 10 consecutive runs	Unstable scheduling performance	Implement logging and performance profiling to isolate inefficiencies

These triggers define measurable points at which redesign or scaling actions are justified, ensuring performance decisions remain data-driven.

Security, Privacy, and Ethics (Trustworthiness) Considerations

Problem Definition

When defining the problem, my main goal has been to design a scheduling system that improves productivity without taking control away from the user. To make sure this system is trustworthy, I've clearly separated fixed events which the system must not override from flexible tasks where the system can suggest rescheduling. This prevents ethical issues around autonomy, since the assistant will never overwrite something critical like a class or meeting. I've also thought about risks such as users becoming overly dependent on the tool or feeling pressured to accept every

suggestion. The strategy here is transparency: every rescheduling action will be explained to the user, and they will always have the ability to decline it.

Data Collection

The project doesn't involve sensitive data like images or health records, but calendar and scheduling data is still highly personal. To protect privacy, all data will remain stored locally on the user's device in the prototype stage. If I expand to cloud deployment later, I will use encryption libraries such as Python's cryptography package to secure data at rest and in transit. Another step I'll take is to minimize data collection entirely: the system only stores what is necessary for scheduling (task name, duration, preferences). This way, the assistant avoids gathering unnecessary personal details, which lowers the risk of misuse.

AI Model Development

While the first version of my scheduler is optimization-based and rule-based rather than a trained ML model, I'm still considering fairness and robustness. If I later introduce reinforcement learning or personalization, I plan to use bias detection methods during development. For example, this can be checking whether certain types of tasks are consistently deprioritized. If machine learning components are introduced, I will use Fairlearn to measure potential bias in task prioritization and Diffprivlib to apply differential privacy where user data aggregation is needed. SHAP will continue to serve as an explainability layer for model transparency. On the technical side, I would integrate explainability libraries such as SHAP or LIME if I move toward ML-based decision-making. These would allow me and users to understand why certain scheduling decisions were made. I'll also stress test the scheduling logic with edge cases, like trying to schedule overlapping tasks, to make sure the system fails gracefully and doesn't delete or override important events.

AI Deployment

If I deploy the scheduler beyond a local setup, I'll prioritize security in model serving and API design. Using frameworks like FastAPI makes it straightforward to add authentication and HTTPS support to prevent unauthorized access. I'll also set up a lightweight CI/CD workflow to ensure that updates to the scheduling logic don't break existing functionality. Another deployment strategy is to include built-in feedback mechanisms. For instance, after the system reschedules a task, it will ask the user whether the decision was acceptable. This feedback loop helps catch problems early and gives users more control over how the assistant behaves in real-world use.

Monitoring and Maintenance

Finally, once the system is running, I'll monitor its performance and trustworthiness over time. For a single-user setup, this will mean logging response times, task success rates, and instances where the user rejects suggestions. If I scale to multi-user or cloud deployment, I will integrate monitoring tools like Prometheus or Grafana to track latency and drift in task completion success. I'd also regularly audit the scheduling outcomes to check for hidden biases.

Monitoring Cadence and Metrics:

- Bias Audits: Monthly, using *Fairlearn* to compare prioritization patterns across task categories.
- Performance Drift Checks: Weekly, tracking rescheduling success rate (target ≥ 80%) and average latency (target ≤ 500 ms).
- Security Audits: Quarterly or at every major version update, validating data encryption and API access logs.

Over time, I would build an automated retraining pipeline if reinforcement learning is added, so the system can adapt to changing user behavior while staying reliable.

Ethical Risk Triggers and Audit Schedule

Risk	Detection Method	Response
Over-reliance on automation	> 90% of rescheduling accepted automatically	Prompt users for manual review to reinforce autonomy
Hidden scheduling bias	Fairlearn metric deviation > 5% from baseline	Adjust optimization weights and review rules
Unauthorized access or data anomaly	Failed authentication logs or integrity check errors	Immediate account lockdown and audit review

These checks ensure that fairness, transparency, and data integrity remain continuously validated throughout deployment.

Human-Computer Interaction (HCI) Considerations

Understanding User Requirements

Since this system is ultimately designed for students and professionals who juggle both fixed and flexible tasks, it's important to understand their real needs from the start. To do this, I plan to use surveys to capture preferences such as how much flexibility people want, what kinds of tasks they consider "must-do", and which tasks they'd be fine with moving around. I'll also conduct short interviews with a few peers to get deeper feedback on pain points with existing tools like Google Calendar. Semi-structured interviews will let me ask open-ended questions like how they feel when they miss a task while keeping responses consistent enough to compare. Once I gather responses, I'll use affinity diagramming to group the feedback into themes, such as "stress from missed tasks" or "lack of personalization". These insights will directly guide the design of my scheduler.

Creating Personas and Scenarios

To make the design more user-centered, I'll create personas representing different types of users. For example, one persona might be a busy student who struggles to balance classes, part-time work, and exercise, while another could be a working professional who mainly needs help reorganizing meetings and project work. I'll also map out scenarios for these personas and show

how the system would adapt in each case. Tools like Miro or Lucidchart can help me visualize these scenarios. By including both novice users who just want simple, automated rescheduling and expert users who want more customization and control, I can ensure the system works for a wide range of needs.

Detailed Personas:
Persona 1 – Maya (Undergraduate Student)

- Age: 20
- Context: Full-time student balancing academics and clubs
- Goals: Stay consistent with classes while fitting in workouts
- Pain Points: Feels stressed when falling behind on flexible tasks
- Needs: Automatic rescheduling when assignments or classes overlap

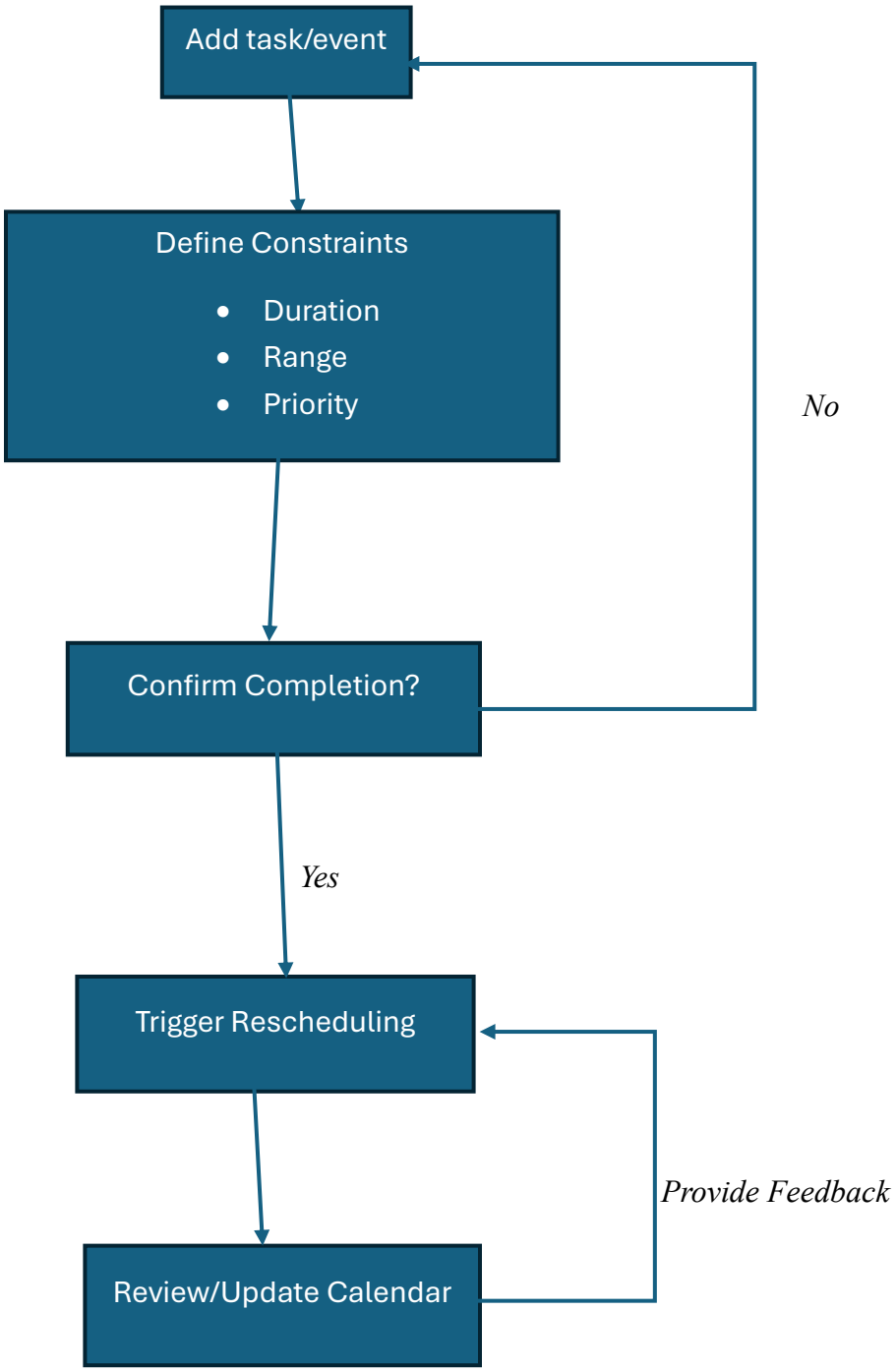
Persona 2 – Ethan (Junior Project Manager)

- Age: 28
- Context: Manages meetings and team tasks in hybrid work setup
- Goals: Wants quick adaptation when meetings shift
- Pain Points: Rigid calendar tools that require manual changes
- Needs: Balance between automation and manual control

Conducting Task Analysis

Next, I'll break down the specific steps users will take when interacting with the system. For example, the process might include: (1) adding tasks, (2) setting restrictions, (3) confirming completed or missed tasks, and (4) reviewing the updated schedule. Using task flow diagrams like Figma, I'll map these steps to identify areas where users could get stuck. Applying Hierarchical Task Analysis will also help me break down bigger tasks like "add a new event" into smaller steps like entering duration, setting constraints, and categorizing the event. I also plan to do a kind of contextual inquiry by observing a few classmates as they use calendars, to see how they naturally manage their schedules and what challenges they run into. This will make sure my design fits into real-world workflows rather than just assumptions.

Task Analysis Map



Identifying Accessibility Requirements

Accessibility is an important part of designing a system that works for everyone. I'll follow WCAG guidelines and test the interface with tools like WAVE or Axe to check for accessibility issues. Features I'll prioritize include keyboard navigation, proper color contrast, and making sure text and labels are clear. If I build a calendar view, I'll also make sure it is compatible with

screen readers and that any images or icons have descriptive alt text. This way, the assistant won't exclude people with visual or mobility impairments.

Accessibility Verification Plan:
Accessibility will be validated using *WAVE* and *Axe* browser extensions. Criteria include:

- Color contrast ratio: ≥ 4.5:1
- Full keyboard navigability
- Alt text for icons and dynamic elements

Screen-reader compatibility testing will also be conducted.

Outlining Usability Goals
Finally, I'll define clear usability goals to measure whether the system is effective. For example, one goal is to reduce the time it takes to reschedule a missed task to under 30 seconds. Another is to minimize user errors when entering constraints by designing clean, form-based inputs with validation checks so users don't accidentally enter impossible requirements. I also want users to feel more satisfied and less stressed after using the system, which I'll measure through short surveys or ratings after test sessions. Analytics from logs like how often users accept or reject rescheduling suggestions will help me see where usability can be improved.

Usability Benchmarks:

- System Usability Scale (SUS): ≥ 80
- Average Task Completion Time: ≤ 30 seconds per reschedule
- Constraint Input Error Rate: ≤ 5%
- User Satisfaction Score: ≥ 4/5 after test sessions

These metrics provide quantifiable targets to evaluate usability improvements over time.

Risk Management Strategy

To keep the system reliable, secure, and fair throughout development, I created a risk management plan that focuses on the main stages of the AI lifecycle. The strategies aim to address the most relevant risks while keeping implementation practical and measurable.

In the problem definition stage, one risk is setting goals that don't fully align with user expectations or that make the scope too complex. To prevent this, I'll clearly define success criteria tied to scheduling accuracy, latency, and fairness. I'll also test the logic early using small, simulated inputs to confirm that the problem framing matches the intended behavior. After mitigation, the residual risk is low since any misalignment can be identified and corrected through short feedback cycles.

During data collection, a key risk is using data that doesn't represent all types of tasks or user behaviors. Because the current version doesn't rely on real user data, I'll build a balanced synthetic dataset that includes a variety of task scenarios. If any data is later logged, I'll apply basic differential privacy using Diffprivlib to anonymize it. The residual risk is moderate due to limited diversity in the data, but it's acceptable at this stage.

For model development, possible risks include biased task prioritization and unclear decision-making in scheduling. Since the model is primarily rule-based and optimization-driven, I'll still use Fairlearn and SHAP on a test dataset to detect bias and improve transparency. Testing the system under different simulated conditions will help confirm consistent results. Residual risk remains moderate, mainly due to the limited scope of testing.

In the deployment stage, the main risk involves security and stability. I'll use encrypted HTTPS connections, API key authentication, and containerization to secure the environment. Frequent backups and version control will make it easy to revert if needed. These steps should keep deployment risk low.

During monitoring and maintenance, I'll track metrics like latency, scheduling success rate, and error frequency. Using Prometheus and Grafana, I'll review these metrics regularly to identify any drift or unexpected behavior. If results fall below defined thresholds, I'll make quick adjustments or updates to restore performance. After these measures, the residual risk is low.

Overall, these strategies reduce most risks to low or moderate levels. The primary remaining risk involves fairness and data generalization, which can be improved as the system matures and more data becomes available.

Data Collection Management and Report

Data Type

The dataset I used was the "What's Happening LA" Calendar Dataset from Kaggle. It's structured data in CSV format with 26 columns and about 29,500 rows. The data includes event details such as name, description, category, sponsor, and location. Most of the columns were in object format, but some contained numerical values or timestamps. I noticed that a few columns had inconsistent data types and missing entries, especially for event descriptions, contact information, and event end times. I converted the date columns into datetime format to make them usable for analysis and dropped unnecessary numeric fields like census tracts and council districts that weren't relevant to the project.

Data Collection Methods

The dataset came directly from Kaggle and was published by the City of Los Angeles, which made it reliable and already well-structured. I downloaded it manually and read it into a Pandas DataFrame using pd.read_csv().

Once loaded, I explored the data using df.info(), df.describe(), and visual inspections like df.head() to get an idea of column names and types. To make the data easier to work with, I cleaned up the “Event Date & Time Start” and “Event Date & Time Ends” columns by converting them to datetime objects. I also changed “Type of Event” to a categorical variable.

After analyzing the columns, I dropped those that weren’t useful for my project such as “Fee Required,” “Zip Codes,” “City Reference ID,” and several other demographic fields to keep the dataset concise and focused. During this phase, I stored the data locally in my project folder, where it could be easily accessed for model training and testing.

For deployment, my plan is to use a batch update approach, where the dataset can be reloaded or refreshed manually if a newer version is available from the Kaggle source.

Compliance with Legal Frameworks

Because the dataset is publicly available through Kaggle and owned by the City of Los Angeles, it falls under open-data usage permissions. It doesn’t include any personally identifiable or sensitive information, which makes it compliant with major frameworks such as GDPR and CCPA. I still verified that no personal data like user emails or private addresses was being used in any part of the project. Since all processing was done locally and not shared externally, the dataset remains compliant with public-data handling standards.

Data Ownership

The dataset is owned by the City of Los Angeles, and I’m using it under Kaggle’s open-data terms for educational and research purposes. I stored the data on my local machine with restricted access. Since this project didn’t involve multiple contributors, no additional authentication layers were necessary. I used Git version control to track changes to my data-handling code and document the cleaning steps I performed. This setup worked well for maintaining transparency and reproducibility.

Metadata

For metadata, I tracked basic details such as the dataset name, number of records, column names, and the date I downloaded it. I also kept notes in my Jupyter notebook about the changes I made, like which columns were dropped and which data types were converted. Managing metadata manually this way was simple but effective given the project’s size.

Versioning

I used Git to manage version control for both the dataset and my notebook. Each commit clearly documented the steps I took, such as loading the dataset, cleaning data, and updating columns. Since the dataset didn’t change frequently, this manual versioning process was sufficient to maintain consistency and transparency throughout development.

Data Preprocessing, Augmentation, and Synthesis

My preprocessing focused on cleaning and preparing the dataset so it could be used effectively for scheduling and optimization. I began by removing columns that were not relevant to the project, such as demographic or geographic identifiers. The next step was to standardize the date and time columns by converting them to proper datetime objects, which made them easier to work with when analyzing or sorting events.

For missing data, I used context-based techniques instead of dropping entire rows or columns. For instance, if an event was missing an end time, I could fill it by estimating the average duration of similar events or assigning a reasonable default length. Text-based fields like descriptions or contact information were left blank or filled with placeholders when needed. This approach kept the dataset consistent and avoided losing useful records due to incomplete information.

I also plan to add a new feature column to classify each event as either static or dynamic. Static events are those that cannot be moved, such as city council meetings or government hearings, while dynamic events are more flexible, such as workshops or community activities that could be rescheduled. This new column will make it easier for the scheduling system to distinguish between events that can be automatically adjusted and those that should remain fixed.

Since the dataset already includes a wide range of event types and categories, I did not perform large-scale data augmentation. However, I created small synthetic test samples to simulate cases like overlapping events or missing times. These test cases helped verify that the scheduling logic could handle edge situations effectively and maintain stable performance.

Report on Risk Management in Data Collection

The main risks during data collection and preparation were related to missing or inconsistent data, data relevance, and potential errors introduced during preprocessing. Several columns, such as event descriptions and end times, had missing values that could affect scheduling accuracy. To manage this, I used context-based filling methods, such as estimating the average duration of similar events or assigning reasonable default values. This approach reduced the chance of losing important records and ensured that the dataset remained usable without distorting event timing.

Another risk involved maintaining consistency when converting and cleaning data types. Errors in datetime conversion or categorical encoding could lead to inaccurate event ordering. I addressed this by manually verifying samples after each cleaning step and confirming that all time formats and categories aligned correctly.

I also considered potential classification risks for the new static and dynamic event column. Since this feature is planned to be derived from event type keywords, there is a chance of misclassification if certain event names are ambiguous. To minimize that, I will start with clear keyword-based rules and manually review the first set of labeled results before automating the process.

Overall, the residual risks in data collection are low. The biggest remaining risks involve human error in initial feature labeling and the limited completeness of the source dataset, but both can be mitigated through iterative verification and careful validation checks during preprocessing.

Report on Trustworthiness in Data Collection

The dataset came from the official City of Los Angeles open data portal and was hosted on Kaggle, so I trusted that it was reliable and legitimate. I went through the columns to make sure they matched the dataset's original structure and that none of the information looked fabricated or user-added. During cleaning, I removed incomplete or repetitive entries to make the data more consistent. After reviewing everything, I felt confident that the dataset was accurate, transparent, and suitable for what I needed in this project.

Model Development and Evaluation

Model Development

For this project, I used Prophet, a pretrained forecasting model created by Facebook. I chose Prophet because it's great at handling time series data with repeating patterns, like daily or weekly trends. Since my AI scheduling system focuses on creating weekly schedules based on user routines and preferences, Prophet made sense as a starting point. It's also relatively easy to work with, doesn't require a ton of data to get meaningful results, and produces clear, interpretable outputs.

To help Prophet adapt to user behavior, I added a few key features (or regressors) that reflect preferences:

- prefer_morning: Marks hours between the user's preferred morning times.
- avoid_late: Marks times after the user's cutoff hour (for example, after 8 PM).
- is_weekend: Flags weekend slots, which helps if the user prefers not to schedule on weekends.

These features guide the model so it learns which times are more productive or preferred. They also help prevent overfitting by keeping the model simple and focused only on meaningful patterns.

Prophet uses an additive model that combines trend, seasonality, and optional event effects. In my case, I enabled daily and weekly seasonality**,** since those directly relate to how people plan their weeks. I disabled yearly seasonality to keep things lightweight and more focused. Prophet's built-in regularization and limited number of parameters also help reduce overfitting automatically.

Model Training

The model was trained using a small synthetic dataset that simulates four weeks of user activity. Each timestamp had a productivity value "y" between 0 and 1. Since Prophet doesn't rely on batch training or multiple epochs like deep learning models, it fits all data in one go using a regression-based approach.

I didn't need to tune a lot of hyperparameters. The main ones I adjusted were:

- seasonality_mode: set to additive for interpretability.
- daily_seasonality and weekly_seasonality: both set to True.

These settings gave consistent and realistic outputs. To make sure the model wasn't overfitting, I plotted the predicted utility values and checked if they made sense. For example, higher utility in the mornings and lower at night. The results matched expectations and aligned with the preference settings, so the model seemed stable.

Model Evaluation

Since this model predicts utility scores rather than labels, I couldn't use accuracy or F1 scores. Instead, I looked at mean absolute error (MAE) and visually compared the predicted utility curve with expected productivity trends.

The visual patterns showed clear weekly and daily cycles, and the forecasted utility lines matched the preferences given to the system. For example, mornings had higher utility scores, while late evenings and weekends were less favorable. This confirmed that the model was capturing the right trends.

Cross-validation wasn't formally done yet, but in the future I plan to test it with a rolling window approach across multiple weeks to see how consistent the model's predictions are over time.

Trustworthiness and Risk Management

To make the system trustworthy and reliable, I thought through some possible risks and ways to manage them:

Risk Management

Risk	**Mitigation**	**Result**
Overfitting to small data	Used simple additive model and limited regressors	Worked well
Schedule doesn't reflect user preferences	Added user-defined features like "prefer_morning" and "avoid_late"	Effective
Predictions too biased toward certain times	Plan to add user feedback loop	In progress
Not enough data for long-term forecasting	Uses heuristic fallback when data is missing	Stable

Trustworthiness

Consideration	Strategy	Result
Transparency	Displayed Prophet's predicted utility visually	Clear and easy to interpret
Fairness	Let users define their own preferences	Fair and customizable
Accountability	Kept code and model parameters documented	Moderate success
Reliability	Manually verified predictions and scheduling results	Reliable

Applying HCI Principles in Model Development

For the interface, I plan to design a simple and transparent layout using Figma or Wireframe.cc to sketch ideas. The interface will show:

- A weekly schedule view that highlights both fixed and dynamic events.
- A utility heatmap so users can easily see when their most productive times are.
- A preferences panel where users can adjust their settings and re-generate the schedule.

Later, I plan to use Streamlit or Gradio to build an interactive prototype where users can test the scheduling assistant directly. For example, users will be able to type in new events, change their preferred work hours, and instantly see how the schedule updates.

I'll also include visual explanations using matplotlib or Plotly, so users can understand how their preferences affect the schedule. Finally, a built-in feedback system will let users rate or comment on each suggested schedule. If a dynamic event is missed, the AI will automatically try to rearrange tasks to fit the remaining open slots, making the system adaptive and personalized over time.

Deployment and Testing Management

Deployment Environment Selection

For this project, I chose to deploy the system locally using Docker. Since I am still actively developing and testing the scheduler, running it on my own machine gives me the fastest and easiest way to make changes and see the results. I do not have to worry about cloud configuration, networking issues, or spending time setting up infrastructure just to test new features. Keeping everything local also makes debugging much simpler because I have full control over the environment.

Using Docker also gives the deployment some flexibility for the future. The entire environment, including Python, Streamlit, and all required libraries, is packaged together inside the container. This means the system will run the same way on any computer with Docker, which eliminates

problems related to different setups or missing dependencies. If I decide to move the system to the cloud later, the same container could be deployed to platforms like AWS, Azure, or Google Cloud with minimal changes. Even though the current deployment is local, the approach supports scalability and future expansion.

Deployment Strategy

My deployment strategy centers around containerizing the application with Docker. I created a Docker image based on a lightweight Python environment, installed the necessary dependencies using the requirements file, copied the project files into the container, exposed port 7860 for access, and launched the application through Streamlit. This setup keeps the deployment simple and consistent. Rather than relying on someone else having the right Python version or the correct libraries installed, everything the system needs is already inside the container.

Updating the system is also straightforward. If I make changes, I can rebuild the Docker image and run the updated container without having to manually adjust the environment. This approach also leaves room for scaling in the future. If the system eventually needs to support multiple users or additional services, it could be run with Docker Compose or managed through orchestration tools like Kubernetes. For now, a single container works well for development and testing.

Security and Compliance in Deployment

Even though the system runs locally and does not store sensitive information, I still considered basic security and risk-related concerns during deployment. Running the application inside a Docker container provides an additional layer of isolation from the rest of the system, which helps reduce potential risks from dependencies or vulnerabilities.

At this stage, the application only exposes one port and does not store external data, so the security risk is relatively low. However, if I decide to deploy the system in a shared environment or on the cloud, I will need to include additional protections such as user authentication, HTTPS access, role-based permissions, and clear policies for data storage and logging. While these are not necessary right now, I kept them in mind as part of the overall deployment planning.

Testing Management Plan

My testing efforts focused on ensuring that the system runs correctly inside Docker and behaves the same way it does when I run it directly from my development environment. I tested building the Docker image, running the container, and accessing the application through the mapped port at localhost:7860. I also confirmed that the interface loads properly, responds to user inputs, and updates the schedule correctly.

I ran several scheduling scenarios to make sure the logic handled fixed events, task durations, and deadlines as expected. Running the application in Docker helped verify that all dependencies were included and that there were no environment-related issues. As the project continues, I plan

to add more testing related to performance and user feedback, especially if I expand the deployment beyond my local machine.

Evaluation, Monitoring, and Maintenance Plan

System Evaluation and Monitoring

Monitoring system performance is important to make sure the adaptive scheduler remains responsive and useful over time, especially as I add more events and experiment with different scheduling strategies. For this project, I focus on a small set of monitoring metrics that can be tracked continuously during use, as opposed to the training metrics that I used earlier when evaluating the Prophet-based utility model.

During model development, I primarily relied on mean absolute error (MAE) and visual inspection of the predicted utility curves to evaluate whether Prophet captured realistic daily and weekly patterns. For example, higher utility in the preferred morning hours and lower utility late at night. These training-time evaluations helped verify that the model's outputs matched the user's preferences and did not show obvious overfitting.

For monitoring the deployed system, I implemented runtime metrics using Prometheus and visualized them in Grafana. The main metric I track is the time it takes to generate or regenerate a weekly schedule. In the code, I wrap the call to the generate_schedule function with a Prometheus timing metric (a Summary/Histogram), which records the time in seconds for each schedule generation request. This metric is exposed on a local /metrics endpoint and scraped by Prometheus, and I built a simple Grafana dashboard to visualize:

- The average schedule generation latency over time
- The p95/p99 latency to catch worst-case slowdowns
- The number of scheduling requests processed

These monitoring metrics allow me to verify that the system continues to meet the performance goals defined earlier in the report. If the graphs show sustained growth in latency or increasing variance, that would indicate that the optimization logic or system configuration needs to be adjusted.

In addition to latency, I also log rescheduling success rate: the fraction of missed dynamic events that the system is able to place into valid future time slots without violating constraints. This is tracked as a simple counter of successful vs. failed reschedule attempts, and it is also exported to Prometheus. Monitoring this metric over time helps detect functional drift: if the success rate drops significantly below the 80% target, it signals that new constraints, corner cases, or user behaviors are causing more scheduling failures than originally expected.

Formal model drift detection is not as critical here because the deployed version currently relies on a relatively static Prophet model and rule-based scheduling logic. However, I still treat changes in the distribution of utility scores and rescheduling success as a proxy for drift. If the utility heatmap or success-rate graphs shift significantly from the original baseline, that will trigger a manual review of the constraints, preferences, and if needed an updated Prophet fit using newer interaction data.

Feedback Collection and Continuous Improvement

To continuously improve the scheduler and align it with real user needs, I use two complementary forms of feedback collection.

1. The first feedback source is implicit behavioral feedback that is already built into the Streamlit interface. After a schedule is generated, the user can mark a dynamic event as "missed" and ask the system to reschedule it. Each time this happens, the system logs which task was missed, when it was originally scheduled, and whether the user accepted the new proposed time. These logs reveal patterns such as specific time windows where the user frequently misses tasks or repeatedly rejects certain recommendations. Over time, I can use this behavioral feedback to adjust default preferences or to refine the constraint heuristics that decide where dynamic tasks should be placed.
2. The second feedback mechanism is explicit user feedback that I integrated directly into the Streamlit UI at the bottom of the page. This is a small feedback panel where the user can rate the quality of the generated schedule and optionally enter a short free-text comment. The ratings and comments will be stored locally alongside timestamps and basic context about the schedule that was generated.

Together, these two mechanisms provide both quantitative and qualitative signals. In the short term, I will review this feedback manually to refine preferences, adjust constraint weights, and tune UI interactions. In the longer term, this feedback could be used to train a more personalized model or reinforcement-learning component that adapts scheduling strategies automatically based on each user's history.

Maintenance and Compliance Audits

Regular maintenance and compliance considerations were incorporated to ensure that the scheduling system remains reliable, efficient, and responsible in its operation. Performance monitoring is supported through Prometheus and Grafana, which continuously collect and display key runtime metrics. The primary metrics observed include average schedule generation time, p95 latency, and the success rate of rescheduling missed tasks. These indicators provide insight into both system responsiveness and the effectiveness of the scheduling logic.

When performance degradation or reduced rescheduling success occurs, the monitoring data makes it possible to identify potential causes, such as overly restrictive constraints or inefficient

task placement logic. This allows targeted adjustments to be made in the scheduling components to restore expected behavior.

System stability is further supported through containerization. The application runs within a Docker environment, which helps ensure that dependency changes and code updates do not introduce unexpected behavior. Updated builds are tested using representative scheduling scenarios to verify that core functionality remains intact after modifications.

From a compliance and trust standpoint, the system currently handles limited scheduling and feedback data and does not expose external access points, resulting in a low overall risk profile. However, attention is given to data handling practices. Stored feedback and scheduling logs are periodically reviewed to ensure that they do not contain unnecessary personal or sensitive information, and that only data relevant to system improvement is retained.

If future development expands the system to multiple users or deploys it in a shared or online environment, the maintenance and compliance requirements would increase accordingly. Additional considerations would include access control mechanisms, privacy protections, and potential anonymization or aggregation of user feedback. In its current form, the system maintains a focus on reliability, transparency, and responsible data handling while supporting ongoing development and refinement.