

# Bulk Production Scheduler

**Bulk Production Scheduler** is an interactive Streamlit application for planning and adjusting production schedules in a cosmetics manufacturing setting. It generates a multi-stage Gantt chart of production orders (batches) going through Mixing, Transfer, Filling, and Finishing operations. The app integrates an LLM (Large Language Model) to interpret natural language or voice commands, allowing users to modify the schedule by simply typing or speaking instructions (e.g. "delay order 5 by 2 hours" or "swap order 12 with 15").

## Functional Overview

- **Multi-Stage Scheduling:** Automatically schedules each order through four sequential production stages – **Mixing**, **Transfer**, **Filling**, and **Finishing** – ensuring no two orders overlap on the same machine. The schedule is initially built by sorting orders by due date and allocating time on each machine in order.
- **Interactive Gantt Chart:** Visualizes the schedule as a Gantt chart with bars representing each operation. Users can filter which orders, products, or machines are displayed, and adjust the color coding by order, product, machine, or operation type for clarity.
- **Real-Time Adjustments:** Users can adjust the production plan on the fly. Natural language commands (text input or voice) are parsed by an AI to apply changes like postponing/expediting orders or swapping the sequence of two orders. The chart updates instantly to reflect changes, maintaining a feasible schedule (no machine conflicts).
- **Voice and Text Commands:** The app supports direct text input for commands and also includes a microphone widget for voice commands. Voice input is transcribed to text, then interpreted, enabling a hands-free interaction mode for production planners.

## Data Sources

The scheduler uses several data files (in the `data/` directory) to define orders and production resources:

- **Orders Data (`orders.csv`):** A list of production orders with columns for `order_id` (e.g. "ORD-052"), `sku_id` (bulk product identifier), `qty_kg` (quantity in kilograms to produce), and `due_date` (requested completion date). This forms the backlog of batches to be scheduled.
- **Production Lines (`lines.csv`):** Defines the available production lines/machines. Each line has a unique `line_id` and a human-readable `name` (e.g. "Mixing/Processing" for `MIX_1`, "Filling/Capping" for `FILL_1`, etc.), a `type` (MIX, TRANS, FILL, FIN), and a brief `description` of that equipment's role. These lines correspond to the stages every order must pass through.
- **Product Definitions (`vrac_products.csv`):** Contains reference data for each bulk product (SKU), including product family and typical processing rates (mixing, transfer, fill, finish rates in kg/hour). This information can be used to calculate operation durations. (*In the current version, a simplified approach with fixed percentages is used to estimate stage durations for each product.*)

Using the above data, the app generates an initial schedule. For each order, the total production time is estimated based on order size (quantity) and a base throughput. This total time is then split across the four

stages according to predefined percentage allocations for that product type. Each stage is assigned to the corresponding machine/line, and the start times are adjusted so that:

- The first operation (Mixing) of an order starts as soon as the Mixing line is free.
- Subsequent operations start after the previous operation finishes *and* the required machine is available (no overlap on the same machine).
- This continues until all orders are placed in sequence across the timeline.

## LLM Integration and Commands

The Bulk Production Scheduler leverages an AI assistant (via OpenAI GPT) to interpret user commands in natural language. This allows production planners to modify the schedule without manually manipulating data. The supported commands include:

- **Delay/Postpone an Order:** Instruct the system to push an order back by a certain duration. For example, “*Delay order 52 by 4 hours*” will start all operations of order **ORD-052** 4 hours later than originally scheduled (if possible). You can also **advance** an order (start earlier) by using terms like “*advance*” or “*bring forward*” (e.g. “*Advance order 54 by 1 day*” to start **ORD-054** one day sooner).
- **Swap Two Orders:** Swap the positions of two orders in the schedule. For example, “*Swap order 52 with order 53*” will exchange the start times of **ORD-052** and **ORD-053**, effectively giving priority to one over the other. The system automatically adjusts all affected operations to ensure the swap doesn’t cause overlaps on shared machines.

**How it works:** The app uses a combination of rule-based parsing and an LLM for robust understanding. It first normalizes the command (ensuring order numbers like “order 5” become standard IDs like ORD-005) and looks for key patterns (e.g. the word “swap” or “delay” with time units). If a command is straightforward (e.g. “*delay ORD-010 by 2 hours*”), the app’s regex-based parser will catch it. For more complex phrasing, it sends the text to the LLM, which returns a structured JSON payload indicating the intent and parameters. The app then validates the intent (checking that order IDs exist, durations are provided, etc.) and applies the change by adjusting the schedule data. After a successful command, a confirmation (success message) is shown, and the Gantt chart updates to reflect the new schedule.

**Voice Commands:** When using the microphone, spoken instructions are transcribed via the Deepgram API. The transcribed text is then processed exactly as a typed command. This means you can simply say, for example, “*swap order 12 and order 15*” aloud, and the app will perform the swap if possible. A small “Voice Debug” panel in the UI can show the last transcript for troubleshooting.

## Setup and Installation

You can run the Bulk Production Scheduler either on Streamlit Cloud or on a local machine. In both cases, you’ll need API keys for OpenAI (for the LLM) and Deepgram (for voice transcription) if you want to use those features.

### Streamlit Cloud Deployment

1. **Repository & Files:** Push the app’s code to a GitHub repository, including `app.py`, the `data/` folder (with CSV files), `nlp_extractor.py`, and the `requirements.txt` file.
2. **Secrets Configuration:** On Streamlit Community Cloud, set the required secrets:  
3. `OPENAI_API_KEY`: Your OpenAI API key (for GPT command interpretation).

4. **DEEPGRAM\_API\_KEY**: Your Deepgram API key (for voice transcription).
5. **Deploy**: From your Streamlit Cloud account, deploy the app by connecting to the GitHub repo. Streamlit will install dependencies from `requirements.txt` and run `app.py`.
6. **Permissions**: Ensure the app has internet access enabled (required for the API calls to OpenAI and Deepgram).
7. Once deployed, you can share the app URL. The Streamlit app will load with the initial schedule and UI ready for interaction.

## Local Installation and Running

1. **Clone the Repository**: Download or clone the project repository to your local machine.
2. **Install Dependencies**: Make sure you have Python 3.10+ installed. Install the required packages using pip:

```
pip install -r requirements.txt
```

This will install Streamlit, pandas, Altair, OpenAI's Python SDK, and other needed libraries. 3. **Set API Keys**: Export your API keys as environment variables in your shell:

```
export OPENAI_API_KEY="sk-xxx..."      # your OpenAI key
export DEEPGRAM_API_KEY="dg-xxx..."     # your Deepgram key
```

(On Windows, use `set OPENAI_API_KEY=...` in the Command Prompt or define them in your IDE's run configuration.) If you don't have these keys or prefer not to use the LLM/voice features, the app will still run – you can use the text command box with simple phrases that the regex parser can handle, or just use the visualization features. 4. **Run Streamlit**: From the project directory, launch the app:

```
streamlit run app.py
```

1. **Open in Browser**: Streamlit will provide a local URL (usually `http://localhost:8501`) that you can open in your web browser. You should see the Bulk Production Scheduler interface with the Gantt chart.

## Using the App

Once the app is running, here's how you can use the Bulk Production Scheduler:

- **Viewing the Schedule**: By default, the Gantt chart will show all scheduled orders. The X-axis is the timeline (dates and hours) and the Y-axis lists the production lines/stages (Mixing/Processing, Transfer/Holding, Filling/Capping, Finishing/QC). Each bar represents an operation for a specific order. You can hover over bars to see details like Order ID, operation type, start/end times, and due date.
- **Filtering Orders/Machines**: Use the sidebar (click the **Filters**  expander if it's hidden) to adjust the view:

- **Orders:** Limit how many orders are shown (e.g., show only the first 10 orders by due date) via the *Orders* number input.
- **Products:** Use the *Products* multiselect to show only specific product types (SKUs). This filters the chart to orders producing those selected products.
- **Machines:** Similarly, use the *Machines* multiselect to view operations on certain equipment only (e.g., only show the Filling line).
- **Color by:** Choose the coloring mode for the bars. For example, select “Product” to color-code bars by product type, or “Operation” to color-code by stage (all mixing operations one color, all filling another, etc.). By default, the chart colors each order uniquely (Color by **Order**).
- After changing filters or color mode, the chart will update immediately. If no operations match the current filters, you’ll see a message indicating no data.
- **Highlighting Orders:** You can click on a bar (which corresponds to a specific order’s operation) to highlight all operations of that order across the timeline. The selected order’s bars will be fully opaque and others dimmed. Double-click on the chart background to clear the selection.
- **Applying Schedule Changes:** At the bottom of the interface, you’ll find the **Command** section:
- **Text Command Input:** Click into the text box (placeholder text: “*Type: delay / advance / swap orders...*”) and type a command to adjust the schedule. Upon pressing Enter, the app will process the command. If the command is understood and valid, the schedule data updates and the Gantt chart is redrawn to reflect the change. You’ll see a green success message if all goes well (or a red error message if the command couldn’t be applied).
- **Voice Command:** Alternatively, click the circular **Voice** button to start recording a voice command (the icon will change while recording). Click again to stop. The app will transcribe your speech and then process the resulting text command just like a typed input. If transcription fails or is silent, you’ll get a warning or error.
- **Examples of Commands:** You can instruct the scheduler in various ways – see below for examples. The commands are quite flexible in phrasing thanks to the AI parser. After any successful command, the chart re-calculates only the necessary parts of the schedule:
  - If you **delay** an order, all of its operations move forward in time, and any other orders on the same machines that would conflict are automatically pushed out as needed (the app ensures no machine is double-booked).
  - If you **advance** (negative delay) an order, its operations move earlier, but the app will not violate machine availability; effectively, it will start as early as possible without overlapping others. Advancing one order may delay another if they swap places.
  - If you **swap** two orders, the start times of their production sequences are exchanged. The order that was later will take the earlier slot and vice versa. The rest of the schedule shifts accordingly for those two orders, but other orders remain in their relative positions.

## Example Commands

Here are some example commands you can try in the app (in the text input or via voice). The order IDs and times can be adjusted based on your actual data:

- **Delay an order:**
  - “*Delay order ORD-052 by 4 hours*” – pushes order 52’s schedule 4 hours later.
  - “*Postpone order 54 by 1 day 6 hours*” – delays order 54 by 1 day and 6 hours.
  - “*Push order 53 back 90 minutes*” – same as delaying order 53 by 1.5 hours.
- **Advance an order (start earlier):**

- “Advance order 54 by 2 hours” – brings order 54 two hours earlier (if possible).
- “Bring forward order 55 by one day” – starts order 55 one day sooner than initially scheduled.
- **Swap two orders:**
- “Swap order 052 with order 053” – exchanges the positions of orders 52 and 53 in the queue.
- “Switch order 51 and order 52” – another way to swap orders 51 and 52.
- **Natural phrasing examples:**
- “Can you delay order 5 by about 8 hours?”
- “Move order 7 ahead by 2 hours.”
- “Swap order seven with order nine.” (uses words “seven” and “nine”; the AI will interpret these as ORD-007 and ORD-009)

Feel free to experiment with phrasing – the LLM is designed to handle different ways of asking for the supported operations. The **Command / OpenAI Debug** panel in the sidebar can be expanded to see how the system interpreted your last command (showing the parsed intent and any errors). This can be useful for understanding unexpected outcomes.

---

## Bulk Production Scheduler – Functional Description

### Overview

The Bulk Production Scheduler is designed to model the production scheduling process in a cosmetics manufacturing environment. In such an environment, making a batch of a product (e.g., a bulk shampoo base, conditioner base, etc.) involves several distinct operations that must happen in sequence. The scheduler captures this by breaking each order into four main stages and assigning each stage to the appropriate machine or production line. It then plans out the timing of these stages for all orders, considering capacity constraints (one task per machine at a time) and order due dates.

This document provides a detailed functional explanation of how the scheduler works, how it models the order lifecycle through the production stages, and how it allows interactive adjustments using natural language commands via an LLM.

### Production Stages in Cosmetics Manufacturing

Each production order (batch) in cosmetics manufacturing typically goes through the following stages:

1. **Mixing/Processing (MIX):** Combining raw ingredients in large mixing tanks to create the bulk product. This involves loading ingredients, mixing (with agitation, heating, etc.), and sometimes waiting for chemical reactions or cooling. The duration depends on batch size and mixing complexity. In our scheduler, an order’s *Mixing* stage is assigned to the **MIX\_1** machine (the primary mixer). We estimate mixing time based on the batch quantity and a base production rate (e.g., larger batches take proportionally more time). For simplicity, the app may use a percentage of total production time for mixing – for instance, a shampoo base might allocate ~10% of its total processing time to mixing.

2. **Transfer/Holding (TRF):** After mixing, the bulk product is transferred (pumped) to a holding tank or intermediate container. In practice, this stage includes setting up pumps, transferring the product, and perhaps holding it in an intermediary vessel until the filling line is ready. The scheduler represents this with the `TRANS_1` line. Transfer times are usually shorter than mixing or filling times; for example, maybe 5–11% of total time depending on product, or calculated via a pumping rate (kg per hour).
3. **Filling/Capping (FILL):** The bulk product is moved through a filling line where it is dispensed into final packaging (bottles, jars, etc.), then capped. The `FILL_1` machine in our data corresponds to the filling line. The duration of this stage is heavily dependent on the fill rate (how many units per minute, or how many kilograms per hour can be filled) and the total quantity to package. In our simplified model, filling might take around 10–13% of the production time for a batch like shampoo or conditioner base. In reality, this could be computed by `time = quantity / fill_rate`.
4. **Finishing/QC (FIN):** Once filled, products go through finishing steps such as labeling, lot coding, packaging into boxes, and quality checks (e.g., weight checks, visual inspection, possibly chemical QC like pH or viscosity tests). The `FIN_1` line represents this final stage. It typically has some fixed overhead time per batch plus variable time per unit. In the model, finishing is a smaller portion of the timeline (e.g., 5–8% of total time for many products), recognizing that it's relatively quick compared to mixing or filling, especially if multiple bottles are processed in parallel during labeling/QC.

**Timing Determination:** The scheduler currently uses a simplified approach to determine how long each stage takes:

- It assumes a base total processing time proportional to the order size. For example, it might assume that a certain quantity (say 300 kg) of any product would take 1 hour of total processing across all stages (this is a hypothetical calibration).
- For each product (SKU), it has a predefined distribution of that total time across the four stages. For instance, a `VRAC_SHAMPOO_BASE` might allocate 10% to Mixing, 9% to Transfer, 13% to Filling, and 8% to Finishing (summing to 40% – the remainder of the time might be considered idle or cleaning/setup). Another product type like `VRAC_HAIR_MASK` might require a bit more time in mixing and transfer (higher viscosity or more ingredients) and relatively less in finishing.
- These percentages are used to calculate each stage's duration: `stage_duration = base_time * percentage`. Base time itself could be `order_quantity / reference_rate`. In our implementation, we used `base_time = qty_kg / 300` as a way to stretch out the timeline to roughly a week for the sample data. (In a real setting, these rates would come from `vrac_products.csv` – e.g., using a specific `mix_rate_kg_per_h` for more accuracy.)

This abstraction means the schedule is not down to the minute accurate for real factory rates, but it captures variability between products (e.g., Hair Mask takes longer in mixing than a Shampoo of the same size, due to viscosity or complexity).

## Order Lifecycle and Machine Scheduling

**Order Lifecycle:** In this scheduling context, an “order” represents a batch that needs to be produced by a certain due date. The lifecycle of an order through the plant is:

- It begins when **Mixing** starts. Before that, the order is simply waiting to be processed (all orders are pending at time zero of the schedule).
- After mixing is complete, the order immediately enters the **Transfer** stage (assuming the transfer line is free; otherwise it waits until `TRANS_1` is available).

- After transfer, it goes into **Filling**, and then **Finishing**.
- Once the Finishing stage completes, the order is considered finished/closed – at that point it should be ready by its due date for downstream activities (like being sent to packaging or to the warehouse).

Throughout this lifecycle, an order cannot jump ahead or skip stages; the sequence is strict. Also, it cannot have two stages running at the same time (you can't fill if mixing isn't done, etc.).

**Initial Scheduling Logic:** The Bulk Production Scheduler creates a baseline schedule by ordering the tasks in a sensible sequence: - **Due Date Priority:** Orders are sorted by due date (earliest due first). This prioritizes more urgent orders to be processed sooner in general. - **Sequential Staging:** We then iterate through the sorted orders and schedule their stages one by one: - For the **Mixing** stage of an order: we start it at the earliest possible time when the mixing tank is free. If it's the first order, we start at a default start-of-week (e.g., Monday 6:00 AM). For subsequent orders, if the mixer was busy with a previous order until a certain time, the next order's mixing will wait until that time. This effectively queues up mixing tasks back-to-back if needed. - For the **Transfer** stage: this cannot start until two conditions are met: (1) the order's mixing is finished, and (2) the transfer line is free. Often, mixing will be the longer task and finishing it determines when transfer can begin, but if the transfer equipment was still busy with a prior order, our order will wait for it. We take the **max** of (mixing end time, transfer line available time) to determine transfer start. - Similarly, **Filling** starts when the later of (transfer completion, filler machine free) occurs. **Finishing** starts when the later of (filling completion, finishing line free) occurs. - This logic ensures that for each machine type, tasks are never overlapping. The scheduler maintains a record of the **next available time** for each machine. Whenever a stage is scheduled on that machine, it updates the machine's next free time. - The result is an **interwoven schedule** where, for example, while one order is in Filling, the next order might already start Mixing in parallel, since those use different machines. This maximizes utilization by keeping different stages working simultaneously on different orders when possible. However, within the same stage line (say the single filling line), orders form a queue.

**Machine Constraints:** Since each stage has a dedicated machine (in this model, one machine per stage type), an important part of scheduling is respecting these constraints: - If two orders would require the same machine at the same time, one must wait. The scheduler by default lines them up in due-date order. - There's no preemption: once a stage starts on a machine, it runs to completion without interruption. Another order's stage can only begin after the current one finishes. - The scheduler doesn't explicitly model maintenance or downtime, but one could imagine blocking off certain time by inserting dummy orders or adjusting the machine's availability start.

**Output Schedule Data:** The scheduler produces a schedule DataFrame where each row is an operation for an order, with columns such as: - `order_id` – which order this operation belongs to. - `operation` – stage (MIX, TRF, FILL, FIN). - `machine_name` – the human-friendly name of the machine performing it (from `lines.csv`, e.g., "Mixing/Processing"). - `start` and `end` timestamps – when this operation is scheduled to start and finish. - `due_date` – the due date of the order (for reference). - `wheel_type` – the product SKU being produced (sometimes called "wheel type" in the data).

This schedule is visualized as the Gantt chart in the app, and it's the data structure that gets updated if changes are made.

## Real-Time Adjustments and Rescheduling

In a real factory schedule, things often change – a machine might break down, a raw material might be delayed, or a rush order comes in. The Bulk Production Scheduler allows a planner to simulate some of these adjustments in real time using natural language commands.

The two main adjustments supported are delaying an order and swapping two orders:

- **Delaying (or Advancing) an Order:** When you delay an order via a command (e.g., “delay order ORD-010 by 8 hours”), the scheduler will push that entire order’s timeline out by the specified duration. Concretely, it finds all the operations for ORD-010 and adds 8 hours to their start and end times. However, simply moving one order later could create conflicts on machines (e.g., if ORD-011 was originally starting after ORD-010, now ORD-011 might start before ORD-010 on the same machine, causing an overlap). The app addresses this by **repacking the schedule** for any machines affected by the move:
  - After adjusting ORD-010’s times, it looks at each machine line. If any operations overlap (ORD-010’s new times versus others), it shifts the subsequent operations forward in time so that they begin right after the previous one ends. This essentially preserves the queue order on each machine but fills any gaps and resolves overlaps.
  - If the command was to advance an order (negative delay, like “advance by 2 hours”), the same logic applies: the order’s operations shift earlier, and if it now overlaps with a previous order’s operations on a machine, that previous order (and any before it) will be pushed earlier as well in a ripple effect. In practice, the app handles advancing by treating it as a negative delay and adjusting overlaps by moving the advanced order as early as possible (it won’t actually pull in earlier orders later – instead, an advance only works if there was slack time or if swapping effectively).
- **Swapping Orders:** A swap command (e.g., “swap order ORD-005 with order ORD-008”) is like a coordinated delay and advance. The scheduler will take the start time of order 5 and order 8 and exchange them:
  - Order 5’s entire sequence is shifted to start where order 8 did, and order 8’s sequence is moved to where order 5 started.
  - This effectively means order 8 is now earlier in the production queue (if order 5 was ahead of it originally) and vice versa.
  - The swap is implemented by calculating the time difference between the two start times and delaying one order by that difference and the other by the opposite (negative that difference). After that, the same repacking logic ensures their operations don’t overlap improperly with others on each machine.
  - The rest of the schedule for other orders remains in the same relative order; only these two orders have swapped slots. This is useful, for example, if a later order becomes higher priority or if an earlier order can be safely delayed.

**Maintaining Feasibility:** The adjustments above maintain a feasible schedule (no overlaps) because of the careful shifting algorithm. However, it’s important to note: - These are local adjustments. The system does not re-optimize the entire schedule globally when one change is made; it simply moves the specified orders. In a complex real-world scenario, one change might ideally prompt a full rescheduling or at least re-check of all downstream impacts. Our scheduler assumes changes are small and isolates their effects. - If you delay an order beyond its due date, the system will still do it (it doesn’t currently enforce due dates as hard constraints, but you’d visually see that it finishes after the due date line on the chart). It’s up to the user to

interpret the consequences. - If an order is advanced earlier than possible (e.g., there is another order currently occupying the machines in that timeframe), effectively the advance will just cause a swap or no-op. The app won't break the rule of one job per machine, so advancing is limited by machine availability. Practically, "advance" commands might swap order priorities or move an order into unused machine time if there was a gap.

## Role of LLMs in User Interaction

A key innovative aspect of this scheduler is the use of a **Large Language Model (LLM)** to bridge the gap between the user's intent and the scheduling system's actions. Instead of requiring the planner to use a rigid user interface or learn specific command syntax, the app lets the planner simply describe what they want to do in plain English (via text or voice). The LLM's role is to interpret these instructions and translate them into structured actions on the schedule.

Here's how the interaction works:

- When the user submits a command (e.g. "Could we push Order 12 out by half a day, and maybe swap Order 9 with 10?"), the system sends this text to an AI model (OpenAI GPT-based). The prompt given to the model instructs it to act as a command interpreter for production scheduling.
- The LLM analyzes the text and attempts to extract the intended operation. It will output a JSON object indicating what the user wants, for example:

```
{  
  "intent": "delay_order",  
  "order_id": "ORD-012",  
  "days": 0,  
  "hours": 12,  
  "minutes": 0  
}
```

This would mean the model interpreted a "push out by half a day" as a delay of 12 hours for order 12. If the user included a swap in the sentence, it might return intent "swap\_orders" with two order IDs.

- If the LLM is not confident or the instruction doesn't match the supported actions, it might return "intent": "unknown". The app will then know it didn't get a clear instruction and can notify the user that the command wasn't understood.
- The scheduler app then takes this structured output and runs the corresponding function (delay or swap). The LLM itself doesn't change the schedule; it's the translator. All scheduling logic (as described earlier) is implemented in code with functions like `apply_delay()` or `apply_swap()`. The model just helps figure out *which* function to call and with what parameters, based on the user's words.

**Why LLM for this?** Traditionally, to interact with a schedule, a planner might use drag-and-drop Gantt charts or fill in a form. The LLM approach lets the planner be more conversational and efficient. For example, they can say "swap A and B" instead of manually reordering tasks. It also allows combining

requests (though our implementation handles one intent at a time; more complex multi-step suggestions could be an extension).

**Voice Integration:** By adding a speech-to-text layer (Deepgram in this case), the user can literally talk to the scheduler. This is especially useful on a factory floor or in meetings where typing might be inconvenient. The voice command goes through transcription, and then the text follows the same LLM path. The end result is the same JSON intent which triggers schedule adjustments.

**Limitations:** The LLM is powerful but not omniscient. It's constrained to only output the supported intents. It won't, for example, create a new order or change quantities – those are outside its scope. If you ask it something completely off-script ("What's the weather during production?"), it will likely return `unknown`. We've also set the LLM to operate with a deterministic, low-temperature setting to ensure consistent outputs. All AI decisions are logged in the app's debug panel for transparency.

## Conclusion

In summary, the Bulk Production Scheduler provides a simplified yet realistic model of cosmetics production scheduling, capturing the essential flow of orders through various stages on limited equipment. The scheduling logic ensures a feasible plan respecting machine constraints and sequence requirements. What sets this tool apart is the interactive layer: planners can intuitively adjust the plan with natural language, thanks to the integration of an LLM for command understanding and APIs for voice input.

This approach demonstrates how AI can enhance operational tools – by making them more user-friendly and adaptive – without replacing the domain rules and calculations that underlie the manufacturing process. The result is a schedule that not only serves as a plan but can also be dynamically updated in response to real-world changes, through a seamless conversation between the user and the system.

---