Closed-Loop Biochemical Therapeutic System with Dynamic Neural Relational Inference Layer

Abstract

A closed-loop biochemical therapeutic control system is disclosed, integrating a hierarchical rule-based arbitration engine with a dynamic neural relational inference layer to maintain patient-specific homeostasis. The system comprises a multi-analyte sensor array (e.g. 13 modular sensors providing ≥140 parallel measurements)[1] and multiple therapeutic actuator cartridges with embedded metadata (drug class codes, dose limits, incompatibility lists)[2]. A deterministic hierarchical arbiter ranks and resolves inputs by priority (harm severity, time-to-harm) and issues control commands at high frequency (e.g. every 5 seconds)[3]. Overlaid on this is an adaptive AI weighting module that learns individual dose-response relationships in real time, building a live patient-specific coupling model (for example, updating a Jacobian sensitivity matrix via micro-dose probing)[4][5]. This dynamic layer infers and continually refines the patient's biochemical "homeostatic envelope," often revealing personalized thresholds and interactions beyond static population-based norms. The AI-driven adjustments are harmonized with fixed safety layers: hard limits and guardrails override unsafe suggestions (e.g. automatic lockout of any cartridge whose drug class conflicts with an active therapy)[2]. Robust fail-safes, including watchdog-monitored redundant controllers and a manual clinician override tier, ensure that any risk is mediated by immediate fallback to safe states. All sensor readings, control decisions, and overrides are immutably logged in a cryptographically secured audit ledger[6] for compliance and traceability. The result is a self-optimizing closed-loop system that autonomously adapts therapy to an individual patient (or analogously, to an industrial process fluid) in real time, while strictly upholding safety constraints and regulatory requirements.

Field of the Invention

The present invention relates to autonomous and semi-autonomous control systems for maintaining biochemical homeostasis in medical and industrial applications. In particular, it concerns a **multi-layer closed-loop control system** that combines a fixed hierarchical safety arbitration engine with a **dynamic machine-learning weighting layer**. The system is applicable to critical care medicine (e.g. ICU drug infusion control, dialysis, ventilator management) as well as process-fluid management in industrial bioreactors or chemical streams. By layering deterministic safety rules with an adaptive **neural relational inference module**, the invention achieves individualized, real-time regulation of complex multi-variable systems.

Background of the Invention

Maintaining stable biochemical conditions (homeostasis) in a complex system – such as a human patient's physiology or a multi-parameter industrial process – is traditionally challenging. Conventional control methods like simple PID (proportional-integral-derivative) loops react only after a parameter deviation has occurred and are typically hard-wired to single inputs and outputs[7]. Such legacy architectures lack adaptability and often struggle with multi-variable interactions, leading to wasted reagents, frequent manual recalibration, and incomplete compliance records[7]. In high-acuity clinical settings, multiple physiological variables (electrolytes, metabolites, cytokines, etc.) can shift rapidly and unpredictably, and manual or static protocol-based interventions often fall behind these dynamic changes[8]. For example, an ICU patient on extracorporeal support may require simultaneous adjustments of fluids, medications, and ventilation; a change in one variable (such as blood CO_2) can have cascading effects on others (pH, electrolytes), complicating single-loop control. Existing automated systems, like insulin pumps or single-analyte controllers, address only one aspect (e.g. glucose) and do not coordinate across dozens of interdependent parameters.

Furthermore, fixed rule-based systems cannot easily accommodate **patient-specific responses or evolving conditions**. Population-derived dosage algorithms or threshold alarms do not account for a particular patient's unique sensitivity or multi-drug interactions. For instance, a drug dose that is therapeutic on average might overshoot in a patient with renal impairment, whereas another patient might metabolize a drug so fast that static dosing is insufficient. Without an adaptive mechanism, controllers either remain conservative (sacrificing efficacy) or risk overshooting and adverse events.

Another challenge is **safety and compliance** in closed-loop biomedical devices. Any autonomous system must guarantee fail-safe operation: if sensors malfunction or if recommended actions could be harmful, the system needs robust override mechanisms. Regulatory standards (e.g. FDA and EU requirements) demand rigorous traceability of decisions and an ability to audit every action, especially when AI components are involved.

In view of these issues, there is a need for a **closed-loop control system** that can: - Integrate **multiple sensor inputs and multiple therapeutic outputs** in one platform, handling complex inter-variable relationships. - Maintain a strict **hierarchical safety structure** (hard limits, conflict avoidance, manual override) to ensure patient safety at all times. - **Adapt dynamically** to each patient or process by learning the unique cause-effect relationships (e.g. drug dose to biochemical response) in real time, rather than relying solely on preset rules or population data. - Provide comprehensive **auditability and compliance**, including immutable logging and secure update mechanisms, to satisfy medical device regulations and allow clinician oversight.

No existing solution in the prior art teaches or suggests such a combined architecture. Traditional ICU infusion systems are either fully manual or employ simple rule-based algorithms for a single parameter. Industrial process control skids might include basic automation and alarms, but they lack the modular cartridge approach and on-line learning of process dynamics. The present invention addresses this gap by introducing a **multi-layer arbitration engine augmented with a dynamic neural inference layer**, yielding a closed-loop system that is at once **safe, adaptive, and transparent** in operation.

Summary of the Invention

The invention provides a **closed-loop multi-analyte homeostasis system** that harmoniously combines **deterministic safety hierarchy** with **adaptive machine-learning control** to regulate biochemical variables in real time. In one aspect, the system comprises: **(a)** a sensor array capable of simultaneously measuring a wide range of biochemical parameters (preferably at least twenty variables) in a subject or fluid; **(b)** a plurality of replaceable **smart cartridges** or actuation modules, each containing an integrated sensor/actuator (such as a drug infusion pump, dialysis filter, adsorbent, or ventilation control valve) and an electronic memory storing metadata like chemical class identifiers, safe operating ranges, and incompatibility rules; and **(c)** a controller (computing device) programmed to execute a **multi-tier control logic**. The control logic is structured hierarchically such that: **(i)** a **vital override layer** monitors for any life-threatening readings and immediately overrides or disables cartridge outputs if a hard safety limit is exceeded; **(ii)** a **guard-rail layer** enforces interlocking rules (for example, suppressing or locking out any cartridge whose drug-class code appears on the incompatibility list of another active cartridge)[4]; **(iii)** an **optimizer layer** performs routine proportional-integral-derivative (PID) or other feedback control adjustments to keep each measurement within a target range; and **(iv)** a **dynamic training layer** (also referred to as the **μ-tier learning layer**) injects small "probe" doses within a safe sub-range (μ-band) when conditions are stable and uses the resulting response data to update a patient- or process-specific model of the system[4]. By this hierarchy, the system maintains safety first while gradually learning the individualized coupling between interventions and outcomes.

Critically, the dynamic inference layer can be realized through various AI modeling techniques working in concert with the deterministic layers. In some embodiments, the controller's adaptive module constructs a **Jacobian matrix** of partial derivatives representing the sensitivity of each monitored variable to each actuator/cartridge, and iteratively refines this matrix through recursive learning as probe data accumulate[5]. In other embodiments, the adaptive layer includes a **graph neural network (GNN)** or similar relational model that treats each variable and intervention as nodes in a graph and learns the strengths of connections (influences) between them. Yet another embodiment employs a **sequence-based predictive model** (for example, a trained bi-directional LSTM or transformer neural network[9]) that analyzes time-series trends of the variables to forecast future deviations and recommend preemptive adjustments. These approaches are not mutually exclusive – the system may combine a fast-updating simple model (like Jacobian linear approximations) with a slower, more complex model (like a neural net) for validation or backup. The **novel integration** of such models into the real-time loop enables the controller to infer a **"personalized homeostatic envelope"** for each patient or process: essentially, a dynamic map of where each variable should ideally be, and how changes in one affect the others, for that specific context.

The dynamic neural inference layer operates as an overlay to the existing hard-coded hierarchy, meaning it does not replace the safety-critical logic but rather **augments it**. For example, if the learned model suggests that a patient's blood pressure will rise in response to a certain drug dose more than expected, the controller can weight that drug's influence lower in the optimizer layer or adjust the dosing schedule proactively. Conversely, if the model identifies that two interventions have synergistic effects, it can moderate their combined dosage to avoid overshooting. All such adjustments are still subject to the guard-rail constraints (e.g., it cannot

command two incompatible drugs together, and it cannot exceed any preset dose ceilings). In essence, the AI layer provides a **context-specific weighting** to the control outputs, continuously tuned by live data, while the deterministic core ensures those outputs remain within a **safe, rule-bounded envelope**.

The invention also includes comprehensive **safety, audit, and compliance features**. The hardware may incorporate redundant controllers and watchdog circuits so that if a fault is detected in the primary control pathway, a secondary controller or mechanical fallback can assume a safe state[10]. A manual **clinician override** control is provided, allowing medical personnel to instantly suspend autonomous operation and take manual control of therapy if needed – for example, pressing an override button might pause all infusions or fix them at a maintenance rate. In preferred embodiments, this manual override state is **timed or monitored**, such that autonomous control can resume automatically after a certain period or once the override condition has been addressed, ensuring that the system continues life-sustaining functions and doesn't remain indefinitely idle without confirmation. The system's software and connectivity are designed to meet regulatory standards: for instance, any software or algorithm updates are cryptographically signed and verified over-the-air[6], and cannot alter the core safety routines without proper authentication.

Every event in the system – sensor readings, algorithmic decisions, issued commands, any human interventions – is recorded to an **immutable audit log**. In an exemplary implementation, each event is time-stamped, serialized (e.g. in JSON), and cryptographically hashed and chained so that a tamper-evident ledger is created[11]. This audit trail may be implemented on a blockchain or write-once medium, and it ensures full **traceability** in compliance with standards like 21 CFR §820 and ISO 13485. A dedicated read-only interface allows regulators or clinicians to inspect the log (for example, retrieving an HL7/FHIR-formatted event history) without any ability to alter data[12][13]. This level of transparency is crucial when advanced AI logic is making treatment decisions – it enables post hoc analysis to explain why the system took a certain action and provides evidence that no records (and thus no decisions) were maliciously or erroneously changed after the fact[13].

In sum, the invention presents a **multi-layer closed-loop control system** that can be embodied as a modular medical device or process control apparatus. It achieves **real-time adaptive control** of complex biochemical systems by combining a **TraceLoop**-style hierarchical arbitration engine (for robust safety and conflict resolution) with a novel **dynamic weighting layer** (for learning and personalization). The synergy of these components yields a platform that is at once **adaptive to individual needs**, **safe by design**, and **fully auditable**. Medical embodiments can autonomously titrate medications, fluids, and other interventions to keep a patient within a personalized safe zone, while industrial embodiments can optimally regulate chemical or bioprocess parameters with minimal waste and downtime. The following description, drawings, and claims provide further details of the architecture, operation, and various embodiments of this innovative system.

Brief Description of the Drawings

1. **FIG. 1** – *System Architecture:* A block diagram of the closed-loop biochemical therapeutic system hardware and top-level architecture. FIG. 1 illustrates multiple sensor modules (e.g., wearable patches, intravenous chips) communicating via a redundant bus

to a central controller, as well as actuator cartridges mounted in bays and external device links. The figure highlights the modular sensor/cartridge design and data flow from sensors into the controller and from the controller to pumps or other actuators.

2. **FIG. 2** – *Hierarchical Control Integration:* A schematic of the control logic layers and integration points. FIG. 2 depicts the hierarchical arbitration engine (vital safety layer, guard-rail conflict layer, PID optimizer layer, etc.) and shows how the dynamic neural inference layer interfaces with these. Arrows indicate the flow of sensor data through both the fixed-rule path and the adaptive model path, converging to a final command output. Sub-diagrams (FIG. 2A, 2B, etc.) may zoom into the dynamic layer's internal model (e.g., a graph network of variables) and the points at which it can modulate the optimizer or raise alerts.

3. **FIG. 3** – *Real-Time Inference and Control Flow:* A timing or flow diagram illustrating the sequence of operations in real time. FIG. 3 shows cycles of sensor readings, data preprocessing, inference by the AI model, execution of control decisions, and feedback. It also indicates the conditions under which a micro-dose probe is injected (when all variables are within the μ-band) and how the resulting change is measured and fed back into the model. The figure emphasizes the closed-loop nature (sense → decide → act → learn) on a continuous basis.

4. **FIG. 4** – *Risk Mitigation and Failsafe Mechanisms:* A state diagram or layered diagram of the system's safety fallback states. FIG. 4 illustrates the escalation from normal operation to various failsafe modes: for example, local safety loops (per-factor guards), global override states (such as switching to a maintenance drip or bypass mode), up to manual override by a clinician (highest level). An annotated flow is shown where if a critical threshold is violated, the system transitions through these layers (with alarms and lockouts) and later automatically falls back to normal once the condition resolves. Also depicted are watchdog monitoring and redundant controller switchover in case of a fault.

5. **FIG. 5** – *Learning Convergence (Adaptive Model Behavior):* A graphical representation of the dynamic learning process over time. FIG. 5, for instance, plots an example patient-specific Jacobian coefficient (sensitivity of a particular output to an input) as it converges over successive probe cycles. It shows the μ-band boundaries for a variable and how the probe magnitude decreases as the controller's confidence in the learned value grows[5]. Another part of the figure may show the concept of a "homeostatic envelope" in multi-dimensional space – for example, a shaded region indicating the range of two variables (like glucose vs. lactate) where the patient remains stable, and how the model refines this region with more data. This figure underscores that the adaptive layer learns more precise control parameters over time, improving the closed-loop performance.

Detailed Description of the Invention

System Overview and Hardware Architecture

Referring first to FIG. 1, the closed-loop system is built around a **modular sensor and actuator architecture**. In a preferred medical embodiment, the system includes a **sensor array** composed

of multiple detachable sensor modules (patches, probes, inline sensors, etc.), each tailored to specific analytes or signals. For example, one module might be a sweat-based electrochemical sensor patch measuring electrolytes and metabolites, another a microneedle array sampling interstitial cytokine levels, another an inline blood gas analyzer, and so on[14][15]. In one implementation, thirteen hot-swappable sensor modules provide on the order of 140 parallel measurement channels covering electrolytes, glucose, lactate, cytokines, coagulation factors, blood gases, pathogen antigens, and other biomarkers[1]. These modules communicate over a multi-drop digital bus (e.g. I^2C or SPI with dual redundancy) to a central controller, ensuring all sensor data streams are synchronized and available in real time[16][17].

On the therapeutic side, the system features multiple **actuator cartridges** (also referred to as intervention cartridges or smart cartridges) docked into slots on a pump cradle or manifold (FIG. 1). Each cartridge contains a reservoir of a therapeutic or regulatory agent (for instance, a drug solution, anticoagulant, electrolyte concentrate, buffering agent, or even a sorbent or filter media), along with microfluidic pumping elements to deliver the agent at controlled micro-dose volumes. In an example configuration, there may be 5 to 15 cartridge bays, each accepting a standardized cartridge. For instance, a "MicroDose-4X" cluster might hold four small syringe pumps for high-precision infusion of medications or electrolytes, while additional bays hold larger-volume pumps or specialized antidote cartridges[18][19]. There may also be **external actuator links** to devices like a ventilator (for controlling oxygen or CO_2 levels), a dialysis machine (for renal support and toxin removal), or an extracorporeal membrane oxygenation (ECMO) pump[20]. The controller can send commands to these external systems via standard interfaces (e.g. RS-485, CAN bus, or network APIs)[21], effectively integrating them into the closed-loop control domain.

Each smart cartridge is equipped with an **on-board memory** (for example, a small EEPROM or FRAM chip) and potentially a microcontroller for local safety interlocks. The memory stores important metadata about the cartridge's contents and limits. In particular, it records a **cartridge identifier and class code**, the concentration or potency of the agent, the maximum recommended dose rate or total dose (often in mg/kg per 24h for drugs), and an **incompatibility list** of other agent classes that should not be co-administered[2][22]. For instance, a cartridge containing a potassium concentrate might be tagged as incompatible with a cartridge containing calcium in certain situations, or an anticoagulant cartridge might be incompatible with a pro-coagulant antidote cartridge – if one is active, the other should not start. The cartridge memory may also define a **μ-band (micro-band)** for one or more variables – essentially a narrow target range [μ_LOW, μ_HIGH] of a specific sensor reading that this cartridge's agent primarily affects[4]. For example, a cartridge delivering a pH-adjusting fluid might store an optimal blood pH band (say 7.35–7.45) as its μ-band for the pH variable. Additionally, calibration data (sensor calibration curves, pump calibration for volume per stroke) and a **unique cryptographic ID or authentication code** can be stored to prevent counterfeit or incorrect cartridges from being used[23].

When a cartridge is inserted into a bay, the system reads its memory to identify the agent and its parameters. The controller cross-checks all active cartridges for any **conflict**: if the new cartridge's class code appears on the incompatibility list of any currently running cartridge, the system will refuse activation or will lock out one of the conflicting pair[24]. This prevents harmful drug–drug or drug–condition combinations from being executed. If all is well, the cartridge becomes part of the control loop. The system can **hot-swap** cartridges without shutting

down: for example, if a cartridge is removed, a sensor detects the disconnection (e.g. a reed switch or circuit break) and the system can automatically open a bypass flow path to maintain circuit continuity[25]. In the brief interval while a cartridge is out, the controller either holds that channel's output at zero or engages a backup (like a maintenance infusion from another source) to ensure no interruption in critical therapy. When a new cartridge is inserted, it undergoes an authentication and priming sequence (e.g. verifying its cryptographic tag, flushing air, and calibrating the sensor) before it's brought online[26]. This modular design affords flexibility (plug-and-play addition of new therapies) and resilience (fast replacement of spent or faulty cartridges) without requiring a system reboot.

All components are unified by the central **controller**, which may be a microprocessor or system-on-chip with real-time capabilities. In one embodiment, the controller is a low-power ARM Cortex-M or NVIDIA Jetson class device running a real-time OS, ensuring deterministic 5-second or faster control loop cycles[3]. The controller board includes communication interfaces for sensors and actuators, and may host both the rule-based logic and the dynamic inference algorithms (possibly with hardware acceleration for AI computations if needed). Power is supplied by a battery (for portability and uninterrupted operation during patient transport or power outage) complemented by AC mains or an external DC supply for long-term use[27]. For example, a 7.4 V lithium-polymer battery can keep the system running for ≥24 hours at typical duty cycles[27]. All components are preferably housed in a portable unit that can be bedside with the patient or next to the industrial process vessel.

Hierarchical Control Architecture (TraceLoop Arbitration Engine)

With the hardware in place, the **control software/firmware** orchestrates the closed-loop regulation through a **hierarchical arbitration engine** (illustrated in FIG. 2). This engine is a layered decision-making protocol that ensures safety and conflict resolution take precedence over routine adjustments. It can be conceptualized as a ladder of control tiers, where the top rungs handle the most critical conditions and can override anything beneath them, while lower rungs deal with finer adjustments and optimizations[28].

1. Vital Override Layer: At the highest priority is the **vital safety or override layer**. This layer continuously monitors all sensor inputs for any value that crosses a **hard safety limit**. These hard limits are preset threshold values for variables that, if exceeded, indicate imminent danger (for instance, oxygen saturation dropping below a certain %, blood pH below 7.0, or pressure above a certain mmHg). The thresholds may be stored in the controller or even in each cartridge's memory for the variable it influences[29]. If any such limit is breached, the vital override layer immediately takes action independent of other logic. Examples of actions include: shutting off or significantly reducing the output of certain cartridges (e.g. if blood pressure is too low, pause any vasodilator infusions), activating emergency cartridges (e.g. trigger a vasopressor or an alkali infusion if pH is critically low), sounding alarms, and potentially switching the system state (for example, entering a "Safe-Monitor" mode where no active dosing occurs until the condition is resolved)[30]. The vital override is essentially a collection of non-negotiable safety rules, each structured as "If X exceeds limit Y, then enforce action Z." This layer guarantees that no matter what the lower control layers or even the AI might be attempting, certain red lines cannot be crossed without intervention.

2. Guard-Rail (Conflict Arbitration) Layer: Just below the vital overrides is the **guard-rail layer**, which deals with incompatibilities and conflicts between control actions. Even if all variables are within safe ranges, not all possible combinations of cartridge activations are permissible. The guard-rail logic uses the information from each cartridge's incompatibility list to maintain a **conflict graph**[24]. In this directed graph, nodes represent active or pending interventions, and edges represent a known conflict (antagonistic or unsafe interaction) between two nodes. If the optimizer (lower layer) or the AI suggests activating a cartridge that would conflict with one already active, the guard-rail layer **vetoes** that action[24]. For example, if an anticoagulant cartridge is running and the system's optimizer attempts to also start a pro-coagulant drug, the guard-rail will block the second action, recognizing it as a direct conflict. Similarly, if a cartridge has a rule like "do not infuse if patient's temperature is below X", and the current temperature sensor reading violates that, the guard-rail will prevent that infusion. In implementation, this can be done by checking each candidate action against a list of exclusions before execution. The guard-rail layer can also enforce **cumulative dose limits** across cartridges or time: for instance, if a patient is approaching the 24-hour maximum dose of a drug, the guard-rail might scale back or temporarily halt that drug's infusion, regardless of optimizer demands[31]. In short, this layer acts as a **safety filter** and logical gatekeeper, ensuring that the combination of actions remains within a predefined safe envelope.

3. Optimizer (Regulation) Layer: Below the safety interlocks comes the **optimizer layer**, which is responsible for the conventional feedback control – adjusting the rates of infusion, filtration, ventilation, etc., to steer the measurements toward their targets. In many embodiments this is implemented as a set of PID controllers or similar control laws for each variable (or each cartridge). For example, if the blood glucose reading is above the desired range, the insulin cartridge's PID loop will increase insulin delivery; if the mean arterial pressure is below target, the fluid or vasopressor cartridge PID will increase output; and so on. The optimizer can operate at a fixed frequency (e.g. every few seconds) reading the current values and computing new actuator commands. Importantly, in a multi-variable system, the optimizer may also consider cross-couplings in a basic way – for instance, it might have a scheduled priority (tuned by the hierarchy's design) such that more critical variables are adjusted first. The hierarchical nature means if two adjustments would conflict (like one says give fluid which might dilute something, another says give drug that concentrates something), the conflict is sorted out by priority or by the guard-rails above. In the baseline design (TraceLoop engine), each rule or control law has an assigned **priority weight based on harm-severity and time-to-harm** for its variable[32]. This ensures that, for example, a rapidly dropping oxygen level (immediate harm) triggers an intervention before a mildly elevated cholesterol (long-term concern) would. The optimizer layer issues **deterministic commands** meaning that given the same inputs it will always produce the same outputs – this predictability is useful for validation and testing. It typically runs continuously during normal operation, adjusting multiple actuators in parallel (unless limited by guard-rails).

4. Dynamic Inference (Training) Layer: Interfacing with the optimizer is the novel **dynamic weighting or training layer**, which will be described in detail in the next section. Briefly, this layer's function is to **learn and personalize** the control strategy for the specific patient or process. In the strict hierarchy sense, this layer can be considered lower than the optimizer in that it does not directly override safety or core control rules; instead, it subtly **tunes** or **provides additional inputs** to the optimizer based on learned insights. In the original TraceLoop

hierarchy, this was implemented as a **μ-tier "probe and learn" mechanism**[4]. Essentially, when the system is stable (all variables within acceptable ranges or within their μ-bands), the training layer intentionally performs a slight perturbation – a tiny increase or decrease in one actuator (e.g., a brief 5% uptick in a drug infusion) – and observes the effect on one or more sensor readings. This yields a data point (input change ΔU, output response ΔX) that the system uses to update its internal model of the patient's dynamics[5]. Over time, by repeated probing of different channels, the controller builds up a **Jacobian matrix** or similar mapping that quantifies how sensitive each measured variable is to each control action for that specific individual[5]. The training layer thus operates continuously in the background of normal control, but only within safe limits (the μ-band ensures the probe is small enough not to push the variable out of its safe range). If at any point conditions become unstable or a higher layer triggers (e.g., a variable leaves the μ-band and goes out of range), the training/probing halts – higher-priority safety and regulation take over until stability is restored.

The above layers describe the logical structure. In practice, the **controller software** cycles through these checks and decisions rapidly. A control loop cycle may proceed as follows (see FIG. 3 for reference):

- **Step 1:** Read all sensor values (possibly hundreds of data points) and update the system state.
- **Step 2:** Apply the Vital Override rules. If any sensor value violates a hard limit, immediately trigger the corresponding override action (set flags to stop certain pumps, initiate alarm, etc.) and possibly short-circuit the rest of the loop (i.e., skip normal dosing this cycle for safety)[29]. If an override was triggered, the system may remain in a failsafe holding pattern until the variable returns to a safe zone.
- **Step 3:** If no top-tier override, evaluate Guard-Rail conflicts. Check planned actions (from last cycle or scheduled) against the incompatibility graph. Remove or inhibit any action that would cause a forbidden combination[24]. Also check cumulative dose budgets – if a particular cartridge is nearing its 24h dose limit, adjust its allowable rate or schedule now to avoid future violation[31].
- **Step 4:** Compute new control outputs via the Optimizer. For each controllable variable or each cartridge, run the control law (e.g. PID) using the latest sensor data as feedback. This yields a tentative set of adjustments (e.g. increase cartridge A by +0.2 mL/hr, decrease cartridge B by –0.1 mL/hr, etc.). These are still subject to the guard-rails; any that were flagged as conflicts are skipped or zeroed out here.
- **Step 5:** Incorporate Dynamic Inference adjustments. Query the adaptive model to refine the optimizer outputs. For example, the model might suggest that the +0.2 mL/hr on cartridge A is too high for this patient because of an observed strong effect, so it scales it down to +0.1 mL/hr. Or it might predict that a small preemptive dose on cartridge C now could prevent a bigger drop in some variable later, even if C's PID alone wasn't going to act yet (this is akin to feed-forward or predictive control). The controller integrates these suggestions **only if** they do not violate any higher-layer rule. The result is a final command set that is both optimized and personalized.
- **Step 6:** Issue commands to cartridges/actuators. The controller outputs the computed pump rates, valve positions, ventilator settings, etc., to the respective devices. This is

done in a deterministic manner, and often at a fixed frequency (to avoid oscillations or race conditions).

- **Step 7:** If conditions are appropriate (e.g., all controlled variables are currently within their target bands or μ-bands), execute a **probe trial** for learning. This might mean choosing one cartridge (or a subset) and adding a tiny increment to its output for a short duration (for instance, 5% above its current rate for the next 30 seconds). The selection of which cartridge to probe may be round-robin or based on which part of the model has the most uncertainty. The magnitude of the probe is computed as a fraction λ of the allowed range (with λ being small, like $\leq$5%)[33]. If a probe is injected, the system closely monitors the resulting sensor changes (ΔX).
- **Step 8:** Log all actions and results. Each sensor reading, each decision (even intermediate ones), and each output command (including any override flags or probe actions) is packaged into an audit log entry[34][11]. This entry is time-stamped and cryptographically signed or hashed into the immutable ledger as described later.
- **Step 9:** Update the adaptive model with new data. If a probe was performed (from Step 7), use the input/output delta to update the appropriate parameters in the model (e.g., perform a recursive least squares update on the corresponding Jacobian row)[33]. Also, incorporate any other new data points (even from normal control actions) into the model's learning process. For instance, if a routine PID change in one drug coincided with a change in a lab value, the model can treat that as a data point too (though it's uncontrolled, so it might weight it differently). Over time, this keeps refining the accuracy of predictions. If the model's confidence in a certain coupling becomes very high (say the 95% confidence interval of that Jacobian parameter falls below a threshold relative to population variance), the system can reduce the amplitude of future probes on that channel to minimize patient disturbance[35]. In effect, as learning converges, the system becomes less perturbative and more fine-tuned.

This loop repeats continuously, for example every few seconds, thus maintaining a tight real-time control over the patient's state. By structuring it hierarchically, the most crucial safety interventions always take precedence, while the fine-tuning and learning occur in parallel without compromising safety.

Dynamic Neural Relational Inference Layer (Adaptive Weighting Module)

A key innovation of this invention is the **dynamic neural relational inference layer**, which endows the system with the ability to learn and adapt its control strategy based on actual observed responses. Unlike traditional control systems that rely on fixed models or tuning, this layer builds a **live model** of the patient or process – essentially capturing the relationships between variables and interventions – and **updates it continuously**. Several embodiments of this adaptive layer are possible, each offering unique advantages. The following are example implementations of the dynamic weighting module:

- **Jacobian μ-Tier Learning (Recursive Least Squares):** In one embodiment, the controller treats the system as initially unknown or only coarsely known, and it learns a **Jacobian matrix** of partial derivatives

$$\partial X / \partial U$$

by injecting small perturbations. This was outlined earlier as the μ-band probing method. Practically, the controller maintains an estimate of how each control input (U) affects each sensor measurement (X). Initially, it might use population data or conservative guesses for these sensitivities. During operation, whenever a **probe micro-dose** is given (within the safe μ-band for that variable), the resultant change in the sensor reading (ΔX) is recorded[5]. Using a recursive least squares (RLS) algorithm or similar adaptive filter, the system updates the corresponding entry of the Jacobian estimate to better fit the observed $\Delta U \rightarrow \Delta X$ mapping[33]. Over many cycles, the matrix converges toward the true patient-specific dynamics. Notably, the controller also adjusts the probe strategy based on confidence: as certain parameters become well-known, the perturbation magnitude λ can be halved or reduced to avoid unnecessary disturbance[35]. The **μ-tier learning** approach effectively personalizes the control gains – for example, if the Jacobian reveals that the patient's blood pressure is extremely sensitive to fluid bolus U1 but less sensitive to vasopressor U2 than expected, the controller will weight U1's adjustments more cautiously and perhaps rely relatively more on U2 than a generic protocol would. This embodiment is powerful in that it is **real-time and interpretable** (the Jacobian is a straightforward representation of sensitivity) and it directly ties into linear control methods. It can reveal cross-couplings too; for instance, if increasing infusion U3 for one condition also consistently affects another variable X4 (side effect), the Jacobian element linking U3 to X4 will capture that, alerting the controller to account for it.

- **Graph Neural Network (Relational Graph Model):** In another embodiment, the adaptive layer is implemented using a **graph-based AI model** that can infer and represent complex, possibly non-linear relationships among variables. Each sensor variable and each actuator could be represented as nodes in a graph, and potential influences as edges. A Graph Neural Network (GNN) model can be trained (either ahead of time on population data and refined online, or entirely online in an unsupervised manner) to predict state changes given certain actions, effectively learning the interaction graph of the patient's physiology. The **Neural Relational Inference** approach uses an encoder-decoder architecture where the encoder learns latent edges (relationships) between entities[36]. Applied here, the encoder would take sequences of multi-variable data and deduce which variables strongly affect which others (for example, detecting that changes in a cytokine level often follow changes in a particular drug infusion, suggesting a causal link). The decoder part of the GNN would then predict forward behavior under different action scenarios. In real-time use, the GNN can output **adaptive weight factors** for each potential action, indicating how "effective" or impactful that action is expected to be on the overall state. For example, if the graph model has learned that a certain patient's inflammatory markers (IL-6, etc.) are highly influenced by fluid status, it might assign a higher weight to interventions affecting fluid balance when trying to reduce inflammation, which a generic protocol might not consider. Conversely, it can suppress or delay actions that it predicts would have negligible effect or would destabilize other connected variables. This GNN embodiment is well-suited to capture **non-linear and multi-step interactions** (like cascades: A affects B which then affects C), which a linear Jacobian cannot fully capture. It essentially **learns the underlying physiology or process dynamics** on the fly, potentially even identifying latent variables (unmeasured factors) through the pattern of interactions. While the GNN is complex, it can run in real

time if kept moderate in size, especially on modern edge computing hardware. Importantly, even the GNN's "belief" about relations can be logged and interpreted to some extent – for example, it might produce an attention matrix highlighting which variables are most connected, which can be visualized for clinician insight.

- **Predictive Sequence Model (LSTM/Transformer):** Another embodiment employs a **time-series prediction model** that learns from the data history to make short-term forecasts of the patient's state. This could be a recurrent neural network like a **Long Short-Term Memory (LSTM)** network or a **Transformer-based model**, as these have been successfully used in sequence prediction tasks[9]. Such a model would ingest a window of recent sensor readings (and knowledge of recent control actions) and output a prediction of future readings (e.g., the next 5 or 10 minutes trend). By training the model (either on prior patient data or on-the-fly with accumulating data, constrained by a validation step to avoid drift[37]), the system gains the ability to **anticipate** problems before they fully manifest. For instance, the model might predict that given the current trajectory, the patient's lactate will rise above the normal range in 30 minutes – something the current value doesn't yet indicate. The controller can use this forecast to take *preemptive action*, such as slightly increasing a drug that improves perfusion or oxygenation to avert the lactate rise[37]. The sequence model thus complements the reactive PID control with a **feed-forward predictive control** element. During normal operation, the controller would periodically query the model: "if we continue current settings, where are we heading?" and if a forecast crosses a threshold (tolerance ε), it will adjust dosing now rather than waiting[37]. Transformers in particular could also handle **multivariate correlations** and seasonal or circadian patterns if present (like daily cycles in certain hormones or metabolic rates). The adaptive aspect here is that the model can be retrained or fine-tuned as more data from the patient becomes available, constantly improving its predictions. Additionally, a **digital twin** validation environment can be used in conjunction: before fully trusting a new model's recommendations, the system might simulate them on past data or known scenarios (as indicated by an offline validation routine)[38], ensuring the model's errors remain within an acceptable bound γ before deployment (to avoid wild, unbounded AI behavior).

Each of these embodiments (Jacobian learning, GNN, predictive model) can be seen as different realizations of the "neural relational inference" concept – they all try to infer the relationships between actions and outcomes. The system could employ one or a hybrid of several, either concurrently or progressively. For example, the Jacobian approach might be used for immediate short-term tuning since it's fast and local, while a GNN might run in the background for deeper insights and only occasionally nudge the control strategy, and the predictive model might primarily serve as an early warning system to the optimizer for upcoming issues.

Crucially, **the dynamic inference layer refines the biochemical homeostatic envelope for each patient in real time.** The term "homeostatic envelope" refers to the range or multidimensional zone within which the patient's physiological parameters are balanced and stable. In a healthy state, homeostasis might correspond to well-known normal ranges (e.g., blood pH ~7.4, lactate <2 mM, etc.), but in a critically ill or highly individualized context, the "normal" for that patient might be different. For instance, a patient with chronic COPD might have a higher baseline CO_2 and lower O_2 saturation that is acceptable for them; or a patient with

traumatic brain injury might require a tighter control of ICP (intracranial pressure) even at the expense of slightly higher blood pressure. The adaptive layer helps identify what that envelope looks like for the current patient. It does so by observing the **combinations of variable values during periods of stability** and noting the patient's reactions to boundary-pushing. If the patient remains stable when a variable is at one extreme of the general normal range but not the other, the system learns that their personal envelope is skewed. If two variables always move together in this patient, the system learns their coupling (e.g., their blood pressure might drop whenever their temperature rises, indicating a unique coupling perhaps due to sepsis; the controller might then treat a fever aggressively to avoid blood pressure dips, which a generic system wouldn't do). Over time, the AI can adjust target set-points within safe bounds. It might tighten the control on a variable if it notices the patient has very little reserve (small envelope) for that variable – for example, if a small drop in oxygen causes a big cascade of issues, it will keep oxygen at the high end of normal for that patient rather than the middle. Conversely, if the patient tolerates a variable's fluctuation without ill effect, the system might allow a wider band for that variable to prevent unnecessary interventions. All these inferences are fed back such that the **optimizer layer's set-points and gains** can be customized: effectively, the adaptive layer whispers to the optimizer "for this patient, prioritize X more, Y less, keep Z around that specific value, and watch out if W goes above this threshold." This synergy ensures the patient's condition is maintained within their personal optimal zone – the true goal of "individualized homeostasis."

Of course, **deterministic safety layers still supervise the adaptive layer.** If the AI were to suggest a target outside clinically accepted safe limits, the higher layers would override that. The design assumes the AI is an aid to performance, not a license to violate safety constraints. In fact, the AI's presence can enhance safety: by predicting and adapting, it may prevent the patient from ever hitting a hard limit in the first place, thus the vital override triggers less frequently. But if an unpredictable event occurs (e.g. a sudden hemorrhage or machine failure), the vital override and guard-rails act instantly, without needing any AI input.

Real-Time Inference and Control Operation

FIG. 3 and the prior descriptions have already touched on the real-time operation, but we will explicitly describe how the dynamic inference and hierarchical control interact on the fly, as this is central to the invention's novelty.

Imagine at time t_0, the system has just been set up for a patient. The sensors are reading baseline values, and the controller has perhaps some default model (maybe from population averages) for expected responses. The clinician has input initial target ranges for key parameters (or they are pre-programmed for typical values). The system starts in a cautious mode: it might run only the deterministic optimizer at first, to stabilize the patient within broad safe limits. During this phase, the dynamic layer is observing passively or maybe doing very tiny probes just to start gathering data.

As the loop continues, the dynamic layer begins to take a more active role. Suppose the patient's blood glucose is slightly above target; the optimizer suggests a small insulin infusion. The dynamic model, based on initial learning, might recall that the last time insulin was given, glucose dropped sharply (perhaps the patient is very insulin-sensitive). It can modify the suggestion: instead of giving the full amount, it proposes 70% of that dose to avoid overshoot. The controller then actually administers that moderated dose. Sure enough, glucose comes down

nicely without going hypoglycemic – the AI prevented an overshoot that might have happened with a one-size-fits-all protocol. This is a simple example of **adaptive dosing**.

Now consider a multi-variable scenario: the patient is on a ventilator and an inotrope drug, and we have sensors for CO_2 (from the ventilator) and blood pressure (affected by the inotrope). These two interventions can conflict because increasing ventilation can lower CO_2 but also reduce cardiac output slightly (affecting BP), while inotrope raises BP but can raise metabolic CO_2 production. A static system might treat them separately or have a fixed coordination rule. Our adaptive system, however, has learned via its GNN or Jacobian that **when ventilation rate is increased, this patient's blood pressure tends to drop significantly** (maybe more than normal). So, when the CO_2 sensor indicates a need to increase ventilation, the dynamic model foresees the BP drop and signals the optimizer to preemptively adjust the inotrope up a bit to compensate. This coordinated move is done in the same cycle: ventilation is increased and simultaneously a compensatory change in drug infusion is made. The patient's CO_2 is corrected without a hypotensive episode, demonstrating how the AI layer **mediates multi-channel interactions in real time**.

Another real-time aspect is **risk detection**. The dynamic model can act as an early warning. For instance, a transformer model might detect a subtle upward drift in inflammatory markers and lactate that historically precedes septic shock in this patient profile. It could alert the controller (and the clinical user interface) that the patient's condition may be deteriorating even before any single variable hits a red alarm. The controller might then tighten certain targets (bring lactate down more aggressively via fluids or adjust ventilation to improve oxygenation) to counteract the trend early. This is a form of **AI-driven prognostic intervention** – going beyond reactive control to anticipatory control.

During all these real-time operations, the system ensures that the **adaptive interventions remain bounded and explainable**. Every time the AI layer alters a dose or recommends a change, the system can log the rationale in terms of data ("Model predicted X would happen, so adjusted Y"). This traceability is important for later auditing and for clinician trust. If an override or guard-rail stops an AI-suggested action, that too is logged ("Action A blocked by guard rule due to conflict with B"). This way, the interplay between the AI and rule-based parts is transparent.

Safety Overrides, Redundancy and Fallback Mechanisms

FIG. 4 depicts the multiple safety fallback paths built into the system. These mechanisms ensure that any failure – whether a sensor fault, a software error, or an unexpected patient reaction – does not result in harm.

Hierarchical Failsafe States: As hinted in the hierarchy, the system can transition to various failsafe modes. For example: - **Safe-Monitor Mode:** If a non-recoverable fault is detected (say multiple sensors fail or a critical calibration is lost), the system can enter a state where all pumps are stopped (or set to a minimum "keep line open" rate) and it just monitors the patient without active intervention[30]. It will signal alarms for human takeover. This is a last resort to "do no harm." - **Bypass Mode:** If a specific cartridge hits a hard safety limit (e.g., a pressure in an extracorporeal circuit too high), the system can isolate that cartridge – closing its line and opening a bypass if available[30]. This prevents that cartridge from causing further issues while allowing others to continue. The affected cartridge is effectively locked out until the condition is

resolved or the cartridge is replaced. - **Maintenance Drip Mode:** For certain critical medications or fluids, a sudden complete stop might itself be dangerous (for instance, stopping a vasopressor abruptly can cause a crash). If the system's battery is critically low or a major fault occurs, it can enter a **maintenance drip mode** where it administers a predefined safe minimal cocktail of essential drugs[39]. This is like a backup program hard-coded for emergencies, providing baseline support (e.g., a bit of vasopressor, fluids, and oxygen) to buy time until the issue is fixed manually. It's akin to how an airplane might go to autopilot holding pattern in an emergency – the patient is kept as stable as possible in a generic way. - **Manual Override:** The highest authority is the human clinician. A physical **override switch or software command** can put the system into a manual mode. In this mode, the clinician can, for example, command specific bolus doses or adjust ventilator settings themselves, using the machine's hardware but bypassing automated decisions. The system, however, doesn't go completely blind even then – it still monitors safety and can intervene if a truly dangerous action is attempted (for instance, if the clinician inadvertently tried to set something beyond a known safety limit, the machine could still block it, or at least warn). The design expects the clinician to use override judiciously; when they do, the event is logged and the system might require a confirmation periodically. For example, an override might automatically expire after, say, 15 minutes unless renewed, to ensure that the system doesn't remain off-line if the clinician becomes busy elsewhere. Once the override period lapses, the system can smoothly transition back to autonomous mode ("fallback" arrow in FIG. 4) if the situation has stabilized[40]. This guarantees that, e.g., after an emergency intervention, the patient continues to get baseline automated care even if the clinician forgets to switch the machine back.

Redundant Hardware and Watchdog: To achieve high reliability (up-time) and meet regulatory demands for critical systems, the device can incorporate redundancy. A common approach included here is a **dual-controller configuration**[10]. Two microcontroller units (MCUs) run in parallel – a primary and a secondary. The secondary either runs the same code in lockstep or at least monitors the primary's outputs. If the primary controller fails (due to a software crash, hardware fault, or it doesn't toggle a watchdog signal in time), the secondary takes over control seamlessly[10]. The transition can be facilitated by hardware logic or by the secondary detecting loss of heartbeat from the primary. In addition, a low-level **watchdog timer** circuit supervises both (and possibly some sensor thresholds too); if neither controller responds correctly, the watchdog can directly trigger a hardware failsafe – for instance, cut power to all pumps (to stop infusions), or switch actuators to a normally safe default (valves might go to a bypass position, etc.)[41]. FIG. 4 (and FIG. 8 in some drawings) show how the data flows from sensors to the primary controller, and in parallel to the redundant controller, with a watchdog overseeing them[10]. This architecture provides **single-fault tolerance**[42], meaning no single point of failure (be it sensor, controller, pump, or power source) will directly cause patient harm without at least one layer catching it.

Cybersecurity and Integrity: The system also includes safeguards against malicious or unintended changes. The firmware and AI models can only be updated via authenticated means[6]. For instance, an update package must be cryptographically signed by the manufacturer or appropriate authority, and the device will verify this signature before applying it (this prevents tampering or unauthorized "hacking" of the control algorithms). The network interfaces are secured, and any remote monitoring or control commands (if allowed) are access-controlled. This is especially pertinent because an AI-driven device could potentially be a target for cyber

attacks – the invention addresses this by leveraging the cryptographic audit chain and secure update pipeline. Even the data being logged is signed to ensure its integrity.

Regulatory Compliance Layer: Implicit in all the above is designing to meet medical device regulations and standards. The system's auditability and safety architecture support compliance with FDA requirements like **21 CFR 820** (quality system regulations) and **21 CFR 11** (electronic records, electronic signatures – the tamper-evident log ensures compliance with Part 11)[11]. The ability to trace every action to either an algorithmic decision or a human input and to prove the record wasn't altered is crucial for post-market surveillance and legal accountability[13]. The device can generate reports or data outputs in standardized formats (HL7/FHIR) for hospital records or regulator review[43]. By implementing an **immutable ledger** of treatment data, the system exceeds typical black-box recorder requirements – it creates a forensic trail that can be audited in case of any incident, which greatly aids in verifying that it operated within its intended parameters[44]. The audit log also has a secondary benefit: it can feed into **future model training and improvement** (with appropriate anonymization). Since all events and outcomes are logged, those data can be retrospectively analyzed to improve the algorithms (the invention even envisions a pipeline for offline model retraining and validation against these logs before deploying new model updates)[45][46]. This establishes a continuous improvement loop under regulatory oversight – new models are checked in a sandbox using historical audit data, signed off, and then disseminated to devices in the field in a controlled manner[47].

Industrial and Process-Fluid Control Embodiments

While the foregoing description has focused on a medical application (a patient in critical care), the invention is equally applicable to **industrial process control** scenarios. Annex Q of the earlier specifications explicitly noted that the same architecture can be applied **mutatis mutandis** to processes involving fluids in bioreactors, chemical manufacturing, water treatment, etc., by substituting the patient and therapeutic context with industrial equivalents[48]. We now outline an example industrial embodiment to illustrate this versatility.

Consider a *biopharmaceutical manufacturing process*, where a 100-liter bioreactor cultivates cells that are sensitive to various environmental parameters (pH, dissolved oxygen, nutrient levels, waste product concentrations, temperature, etc.). Maintaining these within an optimal envelope is critical for yield and quality. Traditional control might involve separate PID loops for pH (adding base/acid), DO (adjusting oxygen sparging), glucose (feeding substrate), etc., and manual interventions if something goes off. Using the present invention, one could deploy a **smart-cartridge skid** controlling the bioreactor inputs. Here, the **sensor array** might include in-line pH and dissolved oxygen probes, redox sensors, turbidity or cell density sensors, perhaps metabolite analyzers (via small sample loops), etc. The **cartridges** would be valves or pumps that add liquids like base, acid, nutrients, anti-foam agents, or remove/process fluid (filtration units to remove waste, for instance). Each cartridge in this case similarly has an ID, class (e.g. "base reagent" vs "anti-foam"), and incompatibilities (you might not want to add two different anti-foam agents together, or certain nutrients might precipitate if mixed incorrectly). The **controller** runs the same hierarchical logic: vital override could correspond to emergency conditions like pH far out of range or a reactor temperature too high, which triggers a shutdown or quench. Guard-rails might include rules like not adding certain combinations of reagents that cause precipitation or not exceeding a certain feed rate to avoid overflow. The **dynamic inference layer** learns the process dynamics – for example, how quickly adding base changes the

pH in this particular reactor given its current state, which might depend on the buffering capacity of the medium and CO_2 levels. Over time, it can refine a model of the reactor's behavior (the process fluid's "homeostatic" or steady-state envelope for optimal cell growth). If the cells' metabolism changes (say oxygen uptake increases as they enter a new growth phase), the adaptive controller will sense that pattern (e.g. DO starts dropping faster than before for the same aeration – indicating a change in oxygen consumption) and will adapt – perhaps by proactively increasing aeration or nutrient feed in anticipation of the next phase, rather than waiting for a drop to occur. It effectively creates a live digital twin of the process.

All safety and logging features remain analogous: every adjustment (valve opening, pump rate change) is logged with timestamp; the data chain is tamper-proof to satisfy auditors or GMP (Good Manufacturing Practice) requirements[11]. If a cartridge (like a nutrient bag) is swapped, the system authenticates it (NFC tag check, etc.) and purges any air before resuming flow[26]. The hierarchical control ensures, for instance, that if an out-of-range pH or redox is detected, the system prioritizes correcting that even if it temporarily means pausing some other optimization (just as in the medical case patient safety came first, here product quality or equipment safety comes first).

One can see that the **structure of claims and features can be read in the industrial context** by simply substituting terms: "human subject" → "process fluid", "therapeutic cartridge" → "processing reagent cartridge", "physiological variable" → "chemical/process variable", etc.[48]. The claims have been drafted to cover both, ensuring broad applicability. For instance, a claim referring to measuring at least twenty biochemical variables in a subject could equally cover measuring at least twenty chemical parameters in a fluid stream; the concept of μ-band (acceptable range for a variable) is the same whether it's blood glucose or bioreactor pH. In industrial use, the dynamic inference might even be easier to validate because one can run repeated batches to train the model. The **novel benefit** in industry is improved efficiency (reduced overshoot means less waste of reagents, tighter control means higher product yield) and easier compliance (the automatic logging replaces manual batch record-keeping).

Another variant could be in **environmental monitoring** or **water treatment plants** where multiple water quality sensors (pH, ORP, turbidity, various ion concentrations) are managed by cartridges adding treatment chemicals (chlorine, flocculants, etc.). The same dynamic control could optimize chemical use while ensuring water stays in potability ranges, and log everything for regulatory records.

In summary, whether in a patient or a process plant, the invention's combination of a **picket-fence hierarchy of safeguards** with an **adaptive learning controller** provides a robust yet flexible solution for complex multi-variable control problems.

Footnotes

---

[1] [2] [3] [6] [8] [14] [15] [16] [17] [18] [19] [20] [21] [22] [27] [32] [42] [52] TRACELOOP OO.docx

file://file_00000000b37071fda52855a6c87fbdab

[4] [5] [7] [9] [11] [23] [24] [25] [26] [29] [30] [31] [33] [34] [35] [37] [38] [39] [48] [49] [50] [51] ProvisionalHierarchyDOCX.docx

file://file-So4BMfLVffY8Pwwxdqtu7o

[10] [12] [13] [28] [40] [41] [43] [44] [45] [46] [47] [53] [54] TRACELOOP OO NEW.docx

file://file-KYjTKHC5fQnnLSfNS42yUo

[36] Dynamic Neural Relational Inference - CVF Open Access

https://openaccess.thecvf.com/content_CVPR_2020/html/Graber_Dynamic_Neural_Relational_Inference_CVPR_2020_paper.html