НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

«На правах рукопису»
УДК 004.89:[616-089.5:615.211]

До захисту допущено:
В. о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«15» травня 2026 р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-науковою програмою «Комп'ютерні науки»
спеціальності 122 «Комп'ютерні науки»
на тему: «Інтелектуальна система визначення препаратів для анестезії при виконанні операції»

Виконав:
студент ІІ курсу, групи КІ-41мн
Копцов Віталій Олегович _____

Керівник:
заст. директора НН ІПСА, д.т.н., професор,
Синєглазов Віктор Михайлович _____

Рецензент:
старший викладач кафедри АСУ, к.т.н.,
НУ «Київський Авіаційний Інститут»
Долгоруков Сергій Олегович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2026 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – другий (магістерський)
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-наукова програма «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
В. о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«30» січня 2026 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Копцову Віталію Олеговичу

1. Тема роботи «Інтелектуальна система визначення препаратів для анестезії при виконанні операції», керівник роботи Синєглазов Віктор Михайлович, д.т.н., професор, затверджені наказом по НН ІПСА від «30» березня 2026 р. №1315-с.

2. Термін подання студентом роботи «15» травня 2026 року.

3. Об’єкт дослідження – процес медикаментозного забезпечення анестезії при хірургічних операціях

4. Вихідні дані – інтелектуальна система здатна визначати тип і кількість анестезіологічних препаратів для введення у реальному часі під час операції

5. Перелік завдань, які потрібно розробити – проаналізувати предметну область анестезіології, види анестезії, основні групи препаратів та ризики їх застосування; дослідити існуючі підходи до застосування штучного інтелекту в анестезіологічній практиці та провести огляд літератури; визначити структуру вхідних даних системи та здійснити постановку задачі; обґрунтувати вибір нейромережевої архітектури та методів навчання моделі; сформувати і підготувати набір клінічних даних, побудувати та навчити

модель; розробити програмне забезпечення системи, провести експеримент і проаналізувати результати.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: електронна презентація

7. Орієнтовний перелік публікацій: основні результати роботи планується апробувати на міжнародній науково-практичній конференції з доповіддю на тему «Застосування архітектури Informer для прогнозування дозування анестетиків на основі інтраопераційних часових рядів».

8. Дата видачі завдання «30» січня 2026 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Огляд досліджень за темою роботи	23.03.2026 – 30.03.2026	Виконано
2	Підготовка першого розділу	31.03.2026 – 05.04.2026	Виконано
3	Підготовка другого розділу	06.04.2026 – 12.04.2026	Виконано
4	Підготовка третього розділу	13.04.2026 – 26.04.2026	Виконано
5	Розробка програмного забезпечення	27.04.2026 – 03.05.2026	Виконано
6	Підготовка четвертого розділу	04.05.2026 – 06.05.2026	Виконано
7	Формування загальних висновків та оцінка перспектив дослідження	07.05.2026 – 10.05.2026	Виконано
8	Оформлення пояснювальної записки	11.05.2026 – 14.05.2026	Виконано

Студент Віталій КОПЦОВ

Керівник Віктор СИНЄГЛАЗОВ

РЕФЕРАТ

Магістерська дисертація: 156 с., 14 рис., 2 табл., 40 посилань, додаток.

АНЕСТЕЗІОЛОГІЯ, ІНТЕЛЕКТУАЛЬНА СИСТЕМА, МЕДИЧНІ ДАНІ, ІНТРАОПЕРАЦІЙНІ ПОКАЗНИКИ, ЧАСОВІ РЯДИ, ГЛИБОКЕ НАВЧАННЯ, ПРОГНОЗУВАННЯ ДОЗУВАННЯ, РЕГРЕСІЙНА МОДЕЛЬ, ПРОПОФОЛ, РЕМІФЕНТАНІЛ.

Об’єктом дослідження є процес медикаментозного забезпечення анестезії при хірургічних операціях.

Предметом дослідження є методи та моделі інтелектуального прогнозування індивідуальних доз анестетиків на основі статичних параметрів пацієнта та динамічних інтраопераційних показників.

Метою роботи є розробити інтелектуальну систему для автоматизованого визначення оптимальних доз анестезіологічних препаратів під час хірургічного втручання.

У роботі досліджено сучасні підходи до прогнозування введення анестезіологічних препаратів під час оперативних втручань та проаналізовано застосування методів штучного інтелекту в анестезіології. Розглянуто основні труднощі прогнозування інтраопераційних часових рядів швидкості інфузії.

Запропоновано та розроблено мультимодульну інтелектуальну систему регресії, у якій як вхідні дані поєднано часові інтраопераційні показники та статичні характеристики пацієнта.

Основну увагу приділено порівнянню різних архітектур кодувальників медичних часових рядів, зокрема LSTM, Transformer та Informer, для прогнозування дозування пропофолу та реміфентанілу. Проведене експериментальне дослідження показало, що модель Informer демонструє найкращу узагальнену якість, забезпечуючи більш точне прогнозування та ефективніше знаходження болюсних введень порівняно з іншими підходами.

ABSTRACT

Master's thesis: 156 p., 14 figures, 2 tables, 40 references, appendix.

ANESTHESIOLOGY, INTELLIGENT SYSTEM, MEDICAL DATA, INTRAOPERATIVE PARAMETERS, TIME SERIES, DEEP LEARNING, DOSE PREDICTION, REGRESSION MODEL, PROPOFOL, REMIFENTANIL.

The object of the study is the process of pharmacological support of anesthesia during surgical operations.

The subject of the study is methods and models for intelligent prediction of individual anesthetic doses based on static patient parameters and dynamic intraoperative indicators.

The aim of the thesis is to develop an intelligent system for automated determination of optimal dosages of anesthesiological agents during surgical interventions.

The thesis investigates contemporary approaches to predicting the administration of anesthesiological agents during surgical procedures and analyzes the application of artificial intelligence methods in anesthesiology. The principal challenges of predicting intraoperative time series of infusion rates are examined, including data irregularity, the presence of noise, and rare peak dosage changes.

A multimodular intelligent regression system was proposed and developed, incorporating temporal intraoperative indicators alongside static patient characteristics as input data.

Particular attention is devoted to comparing different encoder architectures for medical time series namely LSTM, Transformer, and Informer for the prediction of propofol and remifentanil dosages. The conducted experimental evaluation demonstrates that the Informer model achieves superior generalization performance, yielding more accurate predictions and more effective detection of bolus administrations compared to the alternative approaches.

ЗМІСТ

ВСТУП

Сучасна анестезіологія є однією з найбільш відповідальних та технологічно насичених медичних спеціальностей, що безпосередньо визначає безпеку та результати хірургічних втручань. Незважаючи на суттєвий прогрес у зниженні рівня анестезіологічних ускладнень протягом останніх десятиліть, людський чинник у прийнятті клінічних рішень залишається джерелом потенційних помилок. Перспективним шляхом підвищення якості та безпеки анестезіологічної допомоги є розробка інтелектуальних систем підтримки прийняття рішень на основі методів машинного навчання, здатних аналізувати індивідуальні характеристики пацієнта та інтраопераційні дані значно швидше й точніше, ніж це можливо вручну[1].

Об’єктом дослідження є процес медикаментозного забезпечення анестезії при хірургічних операціях.

Предметом дослідження є методи та моделі інтелектуального прогнозування індивідуальних доз анестетиків на основі статичних параметрів пацієнта та динамічних інтраопераційних показників.

Метою роботи є розробити інтелектуальну систему для автоматизованого визначення оптимальних доз анестезіологічних препаратів під час хірургічного втручання.

Для досягнення поставленої мети необхідно розв'язати такі завдання:

- проаналізувати предметну область анестезіології, види анестезії, основні групи препаратів та ризики їх застосування;
- дослідити існуючі підходи до застосування штучного інтелекту в анестезіологічній практиці та провести огляд літератури;

[1] Тут і нижче використано такі інструменти штучного інтелекту як чат-боти з генеративним штучним інтелектом ChatGPT та Claude, виключно для корегування та редагування тексту, створеного автором цієї дипломної роботи, на основі автоматизованої перевірки граматики, структури та стилю, що відповідає Політиці використання штучного інтелекту для академічної діяльності в КПІ ім. Ігоря Сікорського (протокол №11 Вченої ради КПІ ім. Ігоря Сікорського від 11 грудня 2023 р.).

- визначити структуру вхідних даних системи та здійснити постановку задачі;
- обґрунтувати вибір нейромережевої архітектури та методів навчання моделі;
- сформувати і підготувати набір клінічних даних, побудувати та навчити модель;
- розробити програмне забезпечення системи, провести експеримент і проаналізувати результати.

Структура роботи. Дипломна робота складається зі вступу, чотирьох розділів, висновків до розділів, загальних висновків, переліку використаних джерел та додатків. У першому розділі наведено аналіз предметної області, огляд клінічних аспектів анестезіології та існуючих підходів до застосування штучного інтелекту в анестезіологічній практиці. У другому розділі викладено теоретичні основи нейромережевих архітектур та здійснено постановку задачі. У третьому розділі описано побудову інтелектуальної системи, підготовку даних та методи навчання моделей. Четвертий розділ присвячено опису програмного забезпечення, проведенню експерименту та аналізу отриманих результатів.

РОЗДІЛ 1 СУЧАСНІ ПІДХОДИ ДО АНЕСТЕЗІОЛОГІЧНОГО ЗАБЕЗПЕЧЕННЯ ТА ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ ПІДТРИМКИ РІШЕНЬ

1.1 Актуальність задачі автоматизованого підбору препаратів для анестезії

Перша публічна демонстрація загальної анестезії відбулася в 1846 році, але дуже швидко ця процедура стала невід'ємною частиною сучасної хірургічної практики. У світі щорічно виконується близько 310 мільйонів великих оперативних втручань, переважна більшість яких проводиться із застосуванням анестезії. За 70 років з заснування анестезіологічна смертність значно знизилася – з 64 смертей на 100 000 знеболювань у 1940-х роках до 0,4 на 100 000 у 2011 році, що свідчить про суттєвий прогрес у безпеці процедур з розвитком сфери [1]. Однак статистика за 2011 рік стосується переважно пацієнтів без серйозних системних захворювань в розвинутих країнах. У країнах, що розвиваються, показник смертності залишається значно вищим – від 2,4 до 3,3 на 10 000 анестезій. Окремої уваги заслуговують екстрені операції, оскільки на відміну від планових втручань, у таких випадках анестезіолог має діяти швидко і може не мати достатньо часу для детального ознайомлення з історією хвороби пацієнта, його протипоказаннями та індивідуальними особливостями стану. У сукупності ці обставини є однією з причин суттєвого зростання ризику ускладнень та триразового підвищення рівня смертності, асоційованої із загальною анестезією, при екстрених оперативних втручаннях порівняно з плановими.

Вибір типу препарату і рівня введення препарату для анестезії є задачею, що потребує одночасного врахування великої кількості факторів, до яких входять стан пацієнта до і під час операції, його вік, вага, анамнез алергій, супутні захворювання, а також тип, тривалість та терміновость хірургічного

втручання. Існують міжнародні клінічні протоколи і настанови, зокрема розроблені Американським товариством анестезіологів (ASA) та Європейським товариством анестезіології (ESA), які намагаються описати анестезіологчну діяльність, але кінцеве рішення завжди залишається за лікарем і несе в собі ризик людської помилки, зумовленої браком часу, втомою або неповнотою доступної інформації.

Водночас стрімкий розвиток методів штучного інтелекту та машинного навчання, а також можливість збирати і зберігати все більші обсяги клінічних даних, відкривають нові можливості для прийняття анестезіологічних рішень. Використання інтелектуальних систем дозволяє швидко аналізувати історію пацієнта, виявляючи приховані закономірності між клінічними показниками та оптимальною схемою анестезії, що дає можливість підтримувати процес прийняття клінічних рішень і дозволяє значно зменшити рівень людської помилки. Описані вище проблеми разом із сучасними досягненнями у сфері штучного інтелекту зумовлюють актуальність і практичну значущість розробки такої системи.

1.2 Загальні відомості про анестезіологію

Анестезіологія – наука про захист організму від операційної травми та її наслідків, контроль та управління життєво важливими функціями під час оперативного втручання, а також до і після хірургічної операції [2].

Данна робота буде більш за все концентруватися на інтраопераційному періоді, але варто помітити, що сучасна анестезіологія пройшла масштабний шлях трансформації від вузькоспеціалізованої маніпуляційної дисципліни до комплексної галузі періопераційної медицини, яка охоплює всі етапи хірургічного лікування пацієнта. Якщо історично роль анестезіолога

обмежувалася виключно інтраопераційним періодом, то сьогодні професійний обсяг практики включає передопераційну оцінку, оптимізацію стану пацієнта, інтенсивну терапію, управління гострим та хронічним болем [3].

Глобальний обсяг сучасної анестезіології також включає питання медичної інформатики, забезпечення якості та безпеки пацієнтів, а також етичні аспекти догляду в критичних станах. Впровадження цифрових систем моніторингу та електронних медичних карт дозволило систематизувати великі масиви даних, що сприяє прийняттю клінічних рішень на основі принципів доказової медицини.

Перед будь-якою процедурою чи операцією важливо провести детальний огляд медичної історії пацієнта. Сюди входять наявність хронічних захворювань, таких як серцево-судинні або респіраторні патології, попереднє використання ліків та алергічні реакції. Ця інформація допомагає оцінити ризики можливих ускладнень під час процедури.

Найбільш поширеною системою для клінічної оцінки є ASA Physical Status Classification [4] (клас фізичного стану пацієнта ASA), яка допомагає анестезіологу визначити оптимальний план знеболення та моніторинг під час операції. Оцінка враховує тяжкість системних захворювань та функціональні обмеження пацієнта.

ASA клас 1. Пацієнт здоровий, без системних захворювань. Прикладом такого пацієнта є 20-річний здоровий студент-спортсмен запланований на планову операцію з реконструкції передньої хрестоподібної зв’язки. Пацієнт не палить, не вживає алкоголь, не приймає жодних медикаментів.

ASA клас 2. Пацієнт із легкою системною хворобою без значних функціональних обмежень. Прикладом такого пацієнту є 19-річний студент запланований на екстрену операцію після дорожньо-транспортної пригоди. Його індекс маси тіла становить 29, також він повідомляє про рекреаційне вживання психоактивних речовин і добре контрольовану гастроезофагеальну рефлюксну хворобу.

ASA клас 3. Пацієнт із серйозною системною хворобою та значними функціональними обмеженнями. Наприклад 30-річна жінка запланована на планову операцію з видалення великої кісти яєчника. Супутні захворювання включають анемію через менорагію та цукровий діабет II типу, який лікується метформіном. Вона не палить, інколи вживає алкоголь у соціальних ситуаціях і має ІМТ 42. Передопераційна підготовка: ретельне планування анестезії; можливе додаткове лабораторне та кардіологічне обстеження; контроль ліків і стану органів.

ASA клас 4. Пацієнт із тяжкою системною хворобою, що становить постійну загрозу для життя. Приклад: 70-річна жінка запланована на екстрену лапароскопічну апендектомію. Супутні захворювання включають стаж паління 100 пачко-років, погано контрольовану хронічну обструктивну хворобу легень, морбідне ожиріння та інсулінозалежний цукровий діабет II типу. Для такого пацієнта необхідний інтенсивний моніторинг, підготовка до можливих ускладнень під час операції, резервні засоби для стабілізації стану.

ASA клас 5. Пацієнт у критичному стані, який не виживе без операції. Прикладом є 55-річний чоловік запланований на екстрене відновлення великого розриву черевної аорти після того, як його скинув кінь і він напоровся на стовп огорожі. Також під час падіння він отримав 8 переломів ребер і перелом таза. У відділенні невідкладної допомоги у нього сталася зупинка кровообігу через масивну кровотечу, і знадобилося 2 цикли серцево-легеневої реанімації до відновлення спонтанного кровообігу. Наразі він потребує масивного переливання крові та її компонентів через триваючу крововтрату з розриву судини та перелому таза.

ASA клас 6. Пацієнт із констатованою смертю мозку є донором органів для трансплантації. Додаткова передопераційна підготовка включає моніторинг стану донорських органів та координацію з трансплантаційною командою.

Одним із ключових завдань анестезіолога під час загальної анестезії є підтримання оптимальної глибини анестезії, тобто стану, за якого пацієнт не

відчуває болю та не реагує на хірургічні подразники, але водночас не отримує надмірної дози анестетиків. Недостатня глибина анестезії може призводити до інтраопераційного пробудження та стресової реакції організму, тоді як надмірна веде до пригнічення дихальної й серцево-судинної систем.

На ранніх етапах розвитку анестезіології оцінка стану пацієнта ґрунтувалася переважно на клінічних ознаках і спостереженнях лікаря. З метою систематизації цих спостережень у 1937 році доктор Артур Гедель запропонував класифікацію стадій анестезії, яка стала однією з перших стандартизованих систем контролю глибини наркозу та заклала основу для подальшого розвитку методів моніторингу в анестезіології [5].

Стадії анестезії за класифікацією Геделя представлено нижче.

1. Перша стадія – знеболення або дезорієнтація. Відома як «стадія індукції», пацієнт перебуває під седацією, але при свідомості та може розмовляти, дихання повільне і регулярне. Стан переходить від полегшення болю без втрати пам’яті до полегшення болю з амнезією, що закінчується втратою свідомості.
2. Друга стадія – збудження. Характеризується розгальмуванням, неконтрольованими рухами, стійким підвищенням тиску крові та збільшенням частоти серцевих скорочень. Рефлекси дихальних шляхів підвищені, що робить будь-які маніпуляції з дихальними шляхами небезпечними через ризик ларингоспазму та нерегулярного дихання. Для мінімізації часу на цій стадії використовуються швидкодіючі препарати.
3. Третя стадія – хірургічна анестезія. Цільовий рівень для загальної анестезії, що характеризується припиненням рухів очей та пригніченням дихання. Він має чотири рівні. Перший рівень все ще має регулярне спонтанне дихання і націлений погляд, але рефлекси повік та ковтання вже починають зникати на цьому рівні. На другому рівні спостерігається періодична зупинка дихання і втрата

частини рефлексів. Третій рівень описується повним розслабленням м’язів з втратою зіничного світлового рефлексу. Цей рівень вважається ідеальним для більшості операцій, тому він має іншу назву – «справжня хірургічна анестезія». Четвертий рівень характеризується нерегулярним диханням, парадоксальним рухом грудної клітки та повним паралічем діафрагми, що призводить до апное.

4. Четверта стадія – передозування. Виникає, коли доза анестетика перевищує хірургічну стимуляцію, викликаючи сильне пригнічення мозку та кісткового мозку. Він проявляється зупинкою дихання, в’ялістю м’язів, фіксованими розширеними зіницями та критично низьким артеріальним тиском. Без негайної серцево-судинної та дихальної підтримки ця стадія є смертельною .

З часом було створено нові класифікації глибини анестезії і збільшено спектр можливого моніторингу, з’явилися методи моніторингу таких показників як рівень сатурації крові киснем (SpO_2), рівень вуглекислого газу у видихуваному повітрі ($ETCO_2$), артеріального тиску, ЕКГ, температури тощо.

Основною метою анестезії в інтраопераційному періоді є досягнення втрати свідомості, амнезії, аналгезії, релаксації скелетних м’язів та пригнічення вегетативних рефлексів. При цьому анестезіолог часто відповідає за підтримку прохідності дихальних шляхів, бо під час загальної анестезії пацієнт втрачає захисні рефлекси, потребує інтубації та штучної вентиляції легенів. Іноді може порушитися серцево-судинна функція.

Крім загальної анестезії, анестезіологи забезпечують седацію, або контрольований ступінь пригнічення свідомості, при якому пацієнт може виконувати команди або відповідати на подразники. У протилежність седації, загальна анестезія означає повну втрату свідомості (нечутливість до болю й неможливість самостійного дихання).

Вибір між загальною анестезією, седацією та іншими методами знеболення залежить від типу оперативного втручання, його тривалості, інтенсивності больових подразників і загального стану пацієнта. Для великих та травматичних операцій зазвичай необхідне досягнення повної втрати свідомості й контроль життєво важливих функцій, тоді як при менш інвазійних процедурах може бути достатньо лише часткового пригнічення свідомості та адекватного знеболення. У зв'язку з цим в анестезіології важливо чітко розмежовувати поняття седації, аналгезії та інших станів, що застосовуються під час медичних втручань.

Седація, дисоціація та аналгезія – це різні поняття. Седація дозволяє пацієнту залишатися нерухомим. Аналгезія забезпечує знеболення через центральні або периферичні механізми. Дисоціація викликає стан розділення розуму та тіла. Процедурна седація та аналгезія не є ні загальною анестезією, ні виключно знеболювальним засобом. Натомість, це втручання є індивідуальним підходом, який враховує рівень тривожності та болю пацієнта для досягнення оптимальної седації та аналгезії. Процедурна седація та аналгезія в основному використовуються для неінвазійних та малоінвазійних процедур поза операційною, включаючи відділення невідкладної допомоги, стоматологію, радіологію та шлунково-кишкову ендоскопію [6].

1.2.1 Види анестезії та їх клінічні особливості

Залежно від обсягу та локальності хірургічного втручання, а також стану пацієнта, застосовують різні види анестезії.

Загальна анестезія (наркоз) – це стан оборотного пригнічення центральної нервової системи, що супроводжується втратою свідомості, аналгезією, амнезією та м'язовою релаксацією. Існує два способи її створення :

внутрішньовенна і інгаляційна. Така анестезія потребує більшого догляду зі сторони лікаря, оскільки за неї пацієнт втрачає усі захисні рефлекси, тому необхідна інтубація трахеї або підтримка легеневого вентиляційного режиму. Такий вид анестезії застосовується при великих хірургічних втручаннях [7]. Одніим з негативних наслідків загальної анестезії є довгий період виведення з системи, через що протягом кількох днів пацієнт може відчувати загальну слабкість і сонливість.

Седація – це метод для зменшення дратівливості або збудження шляхом введення седативних препаратів, зазвичай для полегшення медичної або діагностичної процедури. Її застосовують для незначних або коротких втручань, коли місцевого знеболення недостатньо, але загальна анестезія не потрібна. Зменшення дратівливості або збудження шляхом введення седативних препаратів, зазвичай для полегшення медичної або діагностичної процедури [8]. Під час процедури дихання може сповільнюватися, іноді необхідно подавати кисень, але пацієнт не втрачає свідомість повністю, як при загальній анестезії. Після завершення дії препаратів люди зазвичай швидко прокидаються, а побічні ефекти, зокрема нудота, сонливість і головний біль, є менш вираженими, ніж після загального наркозу.

Існує кілька рівнів седації залежно від глибини впливу:

- мінімальна – пацієнт розслаблений, перебуває при свідомості та здатний спілкуватися;
- помірна – виникає сонливість, можливе часткове забування процедури, але все ще реагує на голос і тактильне подразнення;
- глибока – пацієнт спить і майже нічого не пам’ятає, але не є повністю непритомним і здатний реагувати на тільки на болючі подразники.

Вибір рівня залежить від типу процедури, стану здоров’я, віку та індивідуальної реакції організму. Седація відрізняється від інших методів анестезії тим, що впливає переважно на рівень свідомості.

Регіональна анестезія – метод знеболення, ціль якого знеболити значну частину тіла, наприклад нижню половину тулуба, пацієнт при цьому залишатися при свідомості. Вона застосовується, коли місцевого знеболення недостатньо, і вводиться через ін'єкцію або катетер біля нервів у зоні операції. Найчастіше регіональна анестезія використовується під час пологів, бо епідуральна анестезія, яка є методом регіональної де препатат вводять в епідуральний простір хребта, що призводить до блокади больової чутливості при збереженні або незначному пригніченні інших видів чутливості і м'язового тонусу і дає змогу пацієнтці залишатися свідомою та активно брати участь у процесі, забезпечуючи поступове знеболення. Іншим варіантом регіональної анестезії є спінальна анестезія, де препарат вводять прям в спинний мозок, вона застосовується для кесаревого розтину або ортопедичних операцій, наприклад на коліні чи стегні.

Також існують інші види регіональної анестезії, зокрема каудальна, паравертебральна та ТАР-блокада, які використовуються при операціях на грудній клітці, черевній порожнині чи нижніх кінцівках [9]. Регіональна анестезія вважається дуже безпечною та має менше побічних ефектів порівняно із седативними та загальними методами. Порівняно з загальною анестезією пацієнти потребують значно меншої кількості післяопераційного знеболення опіатами.

Місцева анестезія – це метод знеболення, для якого відбувається знеболення невеликої ділянки тіла, при якому пацієнт залишається при свідомості. Вона використовується для простих процедур, таких як біопсія шкіри чи грудей, зашивання глибоких порізів або відновлення переломів. Пацієнт може відчувати тиск, але не біль, а відновлення після процедури відбувається швидко, побічні ефекти трапляються нечасто і зазвичай обмежуються невеликим болем у місці ін'єкції.

Таким чином місцева анестезія дозволяє виконувати багато операцій без загальної анестезії або седативних препаратів. У випадках, коли потрібно

знеболити більшу частину тіла, використовують регіональну анестезію. Седація дозволяє провести майже повне знаболення залишаючи пацієнта при свідомості. А використання загальної анестезії забезпечує повну втрату свідомості пацієнта та контроль над больовими відчуттями протягом усього оперативного втручання.

1.2.2 Основні групи препаратів для анестезії

Препарати для анестезії класифікуються залежно від ефекту, швидкості дії та типу застосування: седація, загальна анестезія, регіональна або місцева анестезія [10].

Опіоїдні/наркотичні анестетики. Фентаніл – неймовірно універсальний анестетик, який використовується перед операцією, під час операції та безпосередньо в післяопераційний період. Загалом, фентаніл використовується для лікування гострого болю, при лікуванні пацієнтів із злоякісними онкологічними захворюваннями та у пацієнтів із хронічними больовими станами. Препарат вводять до операції в діапазоні приблизно 50–100 мкг або з урахуванням маси тіла залежно від типу втручання. Під час операцій використовують болюси та інфузію (зазвичай до кількох мкг/кг/год), а введення припиняють за 30-60 хв до завершення, щоб забезпечити швидке відновлення.

Реміфентаніл – препарат, що використовуються для лікування та/або запобігання болю під час та після операції чи інших медичних процедур. Для індукції під інтубацію препарат вводять безперервною інфузією приблизно 0.5-1 мкг/кг/хв або як початковий болюс близько 1 мкг/кг. Під час підтримки інфузія знижується до 0.4 мкг/кг/хв, а в післяопераційному періоді стартують з 0.1 мкг/кг/хв з повільною титрацією до максимуму близько 0.2 мкг/кг/хв.

Гідроморфон – пероральний опіоїдний препарат, який у сім разів потужніший за морфін, що використовується для лікування помірного та сильного болю. Препарат застосовують для знеболення з інтервалами в кілька годин за потреби, залежно від інтенсивності болю. У вигляді таблеток його приймають повторними дозами кожні 4–6 годин для підтримки аналгезії.

Інгаляційні анестетики. Інгаляційні анестетики вводяться використовуючи мінімальну альвеоральну концентрацію(MAC) [10]. Вона визначається як альвеолярна концентрація леткого анестетика при тиску 1 атмосфера, яка запобігає руху скелетних м'язів у 50% пацієнтів у відповідь на болісний хірургічний подразник.

Десфлуран можна використовувати інгаляційно для спричинення втрати свідомості як до, так і після операції у дорослих. Десфлуран також використовується як підтримуючий анестетик після отримання інших анестетиків під час операції у дітей та дорослих. Рівень MAC залежить від віку: у молодих дорослих вона вища, а з віком знижується (7.3% для 25-річних до 6% для 45-річних). Втрати свідомості та амнезія зазвичай настають біля 2.4%, а хірургічний рівень анестезії досягається в діапазоні 2.5-8.5%.

Ізофлуран використовується для індукції та підтримки загальної анестезії. Його використання дозволяє забезпечити стабільний рівень анестезії завдяки керованості глибини седації та швидкій корекції концентрації. Типова доза для дорослого або дитячого пацієнта повинна становити 1.5-3.0% у вдихуваній концентрації для досягнення хірургічної анестезії протягом 7-10 хвилин. Для підтримки хірургічної анестезії слід використовувати концентрацію 1.0-2.5% із закисом азоту, при використанні лише кисню може знадобитися 0.5-1.0%.

Севофлуран застосовують у передопераційному та інтраопераційному періодах для індукції та підтримки загальної анестезії, що супроводжується втратою свідомості та чутливості до больових стимулів. Мінімальна альвеолярна концентрація для хірургічної анестезії із його застосуванням знижується з віком і залежить від використання кисню або закису азоту. При

додаванні закису азоту необхідна концентрація інгаляційного анестетика додатково зменшується, особливо у пацієнтів літнього віку, у яких загальні потреби в анестезії є нижчими.

Внутрішньовенні анестетики. Етомідат використовується завдяки його дозозалежному снодійному ефекту, який прогресує від легкого сну до непритомності [10]. Цей анестетик корисний для пацієнтів з порушеною серцево-легеневою функцією завдяки мінімальному гемодинамічному впливу та зменшеному пригнічуючому ефекту дихання порівняно з іншими внутрішньовенними анестетиками. Для типових дорослих і дітей введення анестезії внутрішньовенно має становити 0.2-0.6 мг/кг із середньою дозою 0.3 мг/кг з 0.2% розчину. Для підтримки етомідату внутрішньовенно слід використовувати менші збільшення дози, ніж початкова; зазвичай дози 0.05-0.1 мг/кг використовуються за потреби із середньою загальною дозою до 0.26 мг/кг.

Кетамін використовується перед операцією та під час операцій, а також широко використовується для запобігання болю та дискомфорту під час певних медичних процедур. При внутрішньом'язовому введенні індукційна доза повинна становити 6.5-13.0 мг/кг, де 9-13 мг/кг внутрішньом'язово забезпечує 12-25 хвилин хірургічної анестезії. Для підтримки дозування слід коригувати відповідно до потреб пацієнта в анестезії.

Метогекситал – ін'єкційний анестетик, який уповільнює активність мозку та нервової системи. Індукція для типової дорослої людини повинна становити 1.0-1.5 мг/кг 1% розчину внутрішньовенно зі швидкістю 1 мл/5 секунд, що зазвичай триває 5-7 хвилин. Періодичні внутрішньовенні ін'єкції 20-40 мг 1% розчину кожні 4-7 хвилин або введення 3 мл 0.2% розчину внутрішньовенно щохвилини.

Пропофол – анестетик, що використовується для уповільнення активності мозку та нервової системи. Він використовується перед операцією для присипляння пацієнтів, а також для підтримки їхньої непритомності під час

загальної анестезії перед операцією чи іншими медичними процедурами. Індукційна доза для типового дорослого віком до 55 років має становити 2-2.5 мг/кг внутрішньовенно, титруючи приблизно 40 мг/10 секунд до початку індукції. Для підтримки типового дорослого віком до 55 років слід використовувати 100-200 мкг/кг/хв внутрішньовенно з 60-70% закисом азоту.

Нейром'язові блокатори (регіональні) анестетики. Цизатракурій зазвичай вводять перед загальною анестезією під час підготовки пацієнта до операції, щоб забезпечити йому нерухомість під час процедури. Цей анестетик також розслабляє горло, що дозволяє легше ввести дихальну трубку перед початком операції. Для дорослих індукційна доза має становити 0.15-0.2 мг/кг внутрішньовенно з додатковою підтримуючою дозою 0.03 мг/кг через 40-65 хвилин. Безперервна інфузія має становити 1-3 мкг/кг/хв [10].

Рокуроній використовується з такими ж цілями як і цизатракурій. Початкова доза перед інтубацією має становити 0,6 мг/кг внутрішньовенно з підтримуючими дозами 0.1, 0.15 та 0.2 мг/кг. Безперервна інфузія має здійснюватися зі швидкістю 10-12 мкг/кг/хв.

Сукцинілхолін використовується для короткочасного розслаблення скелетних м'язів під час медичних процедур після загальної анестезії. Крім того, його використовують для полегшення інтубації з метою розслаблення м'язів горла. Для коротких хірургічних процедур у дорослих пацієнтів слід використовувати дозу 0.3-1.1 мг/кг внутрішньовенно, тоді як для тривалих хірургічних процедур у дорослих пацієнтів слід використовувати дозу 2.5-4.3 мг/хв внутрішньовенно.

Мідазолам використовується для заспокоєння пацієнтів, які перенесли незначні операції, стоматологічні роботи або інші специфічні медичні процедури.

Незважаючи на значний прогрес у безпеці анестезіологічної практики, ризик ускладнень залишається реальним і клінічно значущим. Ускладнення можуть виникати внаслідок індивідуальної непереносимості або алергічної

реакції на препарат, неправильно розрахованої дози, взаємодії між одночасно введеними засобами, а також через недооцінку вихідного стану пацієнта.

Особливу небезпеку становлять рідкісні, але критичні стани злоякісна гіпертермія при застосуванні певних інгаляційних анестетиків, анафілактичний шок при введенні м'язових релаксантів, або глибока депресія дихання при передозуванні опіоїдів. Ризик ускладнень суттєво зростає у пацієнтів із супутніми серцево-судинними, ендокринними або нирковими захворюваннями, де фармакокінетика препаратів може бути суттєво змінена.

1.3 Клінічні протоколи та стандарти вибору анестезії

Розвиток анестезіології супроводжується поступовим формуванням системи клінічних протоколів і стандартів, покликаних уніфікувати підходи до знеболювання та підвищити безпеку пацієнтів. Клінічний протокол у контексті анестезіології – основний медико-технологічний документ, яким мають керуватись медичні фахівці в кожній конкретній клінічній ситуації, уникаючи неефективних та помилкових рішень, а обираючи найефективніше втручання щодо пацієнта [11]. Дотримання таких протоколів дозволяє стандартизувати практику в різних медичних установах, мінімізувати розрізненість рішень і забезпечити відтворюваний на різних рівнх і місцях рівень якості надання анестезіологічної допомоги.

Кожна країна має свої організації, які визначають стандарти, але існують дві організаціїї, які мають значний вплив на анестезіологічну практику Американське товариство анестезіологів (ASA) та Європейське товариство анестезіології та інтенсивної терапії (ESAIC). Вони розробляють і регулярно оновлюють клінічні настанови, від передопераційної оцінки стану пацієнта до

ведення специфічних клінічних ситуацій, таких як важка інтубація або анафілаксія під час анестезії.

Класифікація фізичного стану пацієнта за шкалою ASA [4], яка розподіляє пацієнтів на шість класів залежно від наявності та тяжкості системних захворювань є найбільш відомих і широко використовуваним підходом. Її детальний опис було наведено на початку розділу. Ця класифікація вже довго слугує відправною точкою для оцінки анестезіологічного ризику і часто безпосередньо впливає на вибір препаратів, методу анестезії та обсягу моніторингу під час операції. Американське товариство анестезіологів також публікує стандарти базового інтраопераційного моніторингу [12].

ESAIC, у свою чергу, розробляє більш детальні клінічні настанови, які спрямовані не на загальне підсумовування, а на специфічні категорії пацієнтів і типів втручань. Зокрема настанови щодо анестезії у літніх пацієнтів, при серцево-судинних захворюваннях, в акушерській практиці та при амбулаторних операціях. Ці документи [13] не є незмінними і з розвитком медицини створюються нові протоколи і переглядаються вже існуючі, в основному зміни відбуваються базуючись на систематичних оглядах і метааналізах.

Попри високий рівень деталізації та доказовості сучасних клінічних настанов, їх застосування у реальній практиці все ще має певні обмеження. Навіть найбільш сучасні протоколи не здатні повною мірою врахувати індивідуальні особливості кожного пацієнта, зокрема варіації фармакокінетики та фармакодинаміки, генетично зумовлені відмінності метаболізму препаратів або складні комбінації супутніх захворювань. У результаті лікар-анестезіолог часто змушений адаптувати стандартизовані рекомендації до конкретної клінічної ситуації.

Більшість існуючих протоколів розроблялися переважно на основі даних про відносно здорових пацієнтів. Для осіб із множинними специфічними супутніми захворюваннями, наприклад із серцевою недостатністю, цукровим діабетом та хронічними хворобами нирок, чіткі рекомендації щодо вибору

анестетика часто відсутні або є суперечливими. Це створює значну розбіжність клінічних рішень між різними фахівцями і закладами для таких пацієнтів.

Таким чином ключовою обмеженістю клінічних протоколів є їхня статичність. Вони надають загальні рекомендації, але не завжди здатні запропонувати персоналізоване рішення у реальному часі. Особливо у екстрених ситуаціях, коли час обмежений, а клінічна інформація неповна, традиційні протоколи можуть виявитися недостатньо ефективними для оптимального прийняття рішень. Ці обмеження підкреслюють потребу у розвитку інтелектуальних систем, здатних доповнювати стандартизовані протоколи та забезпечувати динамічну і персоналізовану підтримку анестезіологічних рішень.

1.4 Застосування штучного інтелекту в анестезіології

Потенційне практичне застосування штучного інтелекту в анестезіології охоплює всі етапи періопераційного процесу. Основними цілями є зниження навантаження на медичний персонал, забезпечення альтернативної думки професіоналу, допомога в визначенні прихованих властивостей і покращенні індивідуалізації медицинської практики.

Сучасні дослідження спрямовані на аналіз фізіологічних сигналів, прогнозування ускладнень, оцінку глибини анестезії, автоматизацію введення препаратів та оптимізацію клінічних процесів. Залежно від клінічної задачі системи штучного інтелекту використовують як структуровані медичні дані, так і часові ряди фізіологічних параметрів, медичні зображення, відео та текстову документацію.

Штучний інтелект має потенціал революціонізувати анестезіологічну допомогу, покращуючи результати лікування пацієнтів, зменшуючи витрати та підвищуючи ефективність надання анестезії [14].

1.4.1 Передопераційна оцінка пацієнта

Передопераційна оцінка пацієнта є першим етапів анестезіологічного забезпечення пацієнта, оскільки саме на цьому етапі визначаються потенційні ризики проведення анестезії, складність забезпечення прохідності дихальних шляхів та ймовірність розвитку періопераційних ускладнень. Через можливість швидкого та водночас глибокого аналізу штучним інтелектом з’являється можливість аналізувати клінічні дані та формувати персоналізовані стратегії з урахуванням індивідуальних особливостей для кожного пацієнта, який готується до операції.

Одним із досліджуваних напрямів є оцінка складності інтубації та проведення ларингоскопії. Традиційні методи значною мірою залежать від суб’єктивної оцінки лікаря та можуть мати обмежену точність [15]. Системи штучного інтелекту використовують параметри індекс маси тіла, особливості будови шиї та обличчя, а також фотографії й відеозображення пацієнта для автоматизованого прогнозування складної інтубації [16]. Особливо активно розвиваються методи комп’ютерного зору, здатні аналізувати анатомічні особливості дихальних шляхів за зображеннями обличчя та шиї [17, 18].

Іншим важливим напрямом є класифікація пацієнтів за рівнем операційного ризику. Алгоритми машинного навчання використовуються для прогнозування ймовірності післяопераційних ускладнень, госпіталізації до відділення інтенсивної терапії, повторної інтубації, гострої ниркової недостатності, післяопераційної гіпотензії та летальності. Для цього системи

аналізують демографічні характеристики пацієнта, історію захворювань, результати лабораторних досліджень, медикаментозну терапію та фізіологічні показники. Також моделі можуть застосовувати для цього комбінацію преоперативних та інтраоперативних даних.

Окрему увагу приділено автоматизації оцінки фізичного стану пацієнта за класифікацією ASA. Незважаючи на широке клінічне використання, дана система значною мірою залежить від суб'єктивної оцінки анестезіолога [15]. У зв'язку з цим досліджуються підходи, які дозволяють автоматично прогнозувати ASA-статус на основі супутніх захворювань, призначених препаратів та електронної медичної документації. Крім того, сучасні системи здатні не лише виконувати одноразову оцінку перед операцією, а й динамічно оновлювати прогноз ризику впродовж періопераційного періоду на основі нових клінічних даних.

Окремим напрямом є профілактика небажаних побічних реакцій на лікарські засоби. Використовуючи методи машинного навчання та аналіз великих масивів медичних даних, системи ШІ можуть виявляти закономірності, пов'язані з ризиком розвитку ускладнень, та попереджати лікаря про потенційно небезпечні ситуації при застосуванні анестезіологічного препарату ще до їх виникнення [15].

1.4.2 Інтраопераційний моніторинг та підтримка прийняття рішень

Інтраопераційний моніторинг та підтримка прийняття клінічних рішень є напрямком в якому ШІ може значно допомогти спеціалістам, оскільки під час операції анестезіолог повинен безперервно контролювати життєво важливі показники пацієнта, глибину анестезії, реакцію організму на хірургічне втручання та своєчасно коригувати введення препаратів. Інтраопераційний

моніторинг включає безперервну оцінку серцево-судинних, дихальних та нейрофізіологічних параметрів пацієнта. ШІ дозволяє автоматизувати частину цих процесів [14], підвищуючи точність і швидкість реагування на зміни стану пацієнта.

Підхід до проблеми таким чином показує, що ключовим напрямком використання ШІ під час операції є управління дозуванням анестетиків. Для безпечного проведення анестезії необхідне точне введення лікарських засобів. Системи на основі ШІ здатні швидко аналізувати вік пацієнта, масу тіла, супутні захворювання, фізіологічні показники та особливості операції для визначення оптимального дозування препаратів. Такі системи також можуть використовувати дані моніторингу в режимі реального часу для автоматичного коригування швидкості інфузії анестетиків [14].

У сучасній анестезіології активно розвиваються автоматизовані системи введення препаратів та замкнуті системи керування анестезією. Вони поєднують фармакокінетичні та фармакодинамічні моделі з алгоритмами машинного навчання для підтримання стабільної концентрації препаратів і необхідної глибини анестезії [15]. Такі системи функціонують за принципом зворотного зв’язку, коли зміни фізіологічних параметрів пацієнта автоматично впливають на швидкість введення анестетика.

Одним із важливих частин інтраопераційного моніторингу є контроль глибини анестезії. Глибина анестезії визначається балансом між дозою анестетика та рівнем хірургічної стимуляції. Вона має кілька рівнів, що відрізняються за рівнем сприйняття пацієнтом больових подразників. Для її оцінювання використовуються електроенцефалографічні сигнали, показники гемодинаміки та інші фізіологічні параметри. Найбільш поширеним індикатором гибини анестезії є біспектральний індекс. Було розроблено моделі, які змогли перевершити по точності традиційні методи визначення глибини анестезії [15].

ШІ також використовується для підтримки прийняття клінічних рішень під час операції. Інтелектуальні системи можуть аналізувати дані пацієнта та надавати рекомендації щодо вибору анестетиків, корекції доз, підтримки рівня глибини анестезії та післяопераційного ведення пацієнту [14]. Такі системи дозволяють анестезіологу швидше виявляти зміни стану пацієнта у режимі реального часу прогнозуючи можливі ускладнення.

Перспективним напрямом є також застосування ШІ в варіанті регіональної анестезії, яка відбувається під ультразвуковим контролем. Застосування ШІ є можливим, оскільки інформація з апарату є зображенням, на якому системі необхідно розпізнавати нерви та анатомічні структури. Краще визначення цілі анестезії підвищує точність виконання блокад і зменшує ризик ускладнень при цьому скорочуючи тривалість процедур [15].

1.4.3 Післяопераційний догляд та прогнозування ускладнень

Дослідження в цій категорії спрямовані на використання технологій штучного інтелекту для прогнозування післяопераційного стану пацієнта, оцінки потреби в інтенсивній терапії, визначення готовності до виписки та раннього виявлення можливих ускладнень.

Системи штучного інтелекту забезпечують швидкий та ефективний аналіз медичної інформації кожного пацієнта, що дає змогу своєчасно виявляти ознаки погіршення стану та підвищує якість моніторингу без додаткових витрат часу на індивідуальний аналіз [19].

Одним із варіантів використання ШІ є прогнозування післяопераційних ускладнень – гіпотензії, кардіореспіраторних порушень, гострої ниркової недостатності та ризику повторної інтубації [20, 21]. Використання штучного

інтелекту дає можливість раніше визначати пацієнтів із підвищеним ризиком розвитку ускладнень та своєчасно коригувати тактику лікування.

Також пропонуються механізми за допомогою яких ШІ можна бути застосовувати для безперервного післяопераційного моніторингу пацієнтів. Інтелектуальні системи можуть у режимі реального часу здатні аналізувати життєво важливі показники, автоматично виявляючи аномальні зміни та попереджючи медичний персонал про можливі критичні стани [14].

Іншим способом застосування ШІ в післяопераційному догяді можна назвати оцінку та контроль болю. Традиційна оцінка болю значною мірою залежить від суб'єктивних відчуттів пацієнта, тому сучасні розробки інтелектуальних систем спрямовані на створення більш об'єктивних методів оцінювання. Для цього використовується аналіз поведінкових реакцій, міміки, дихання та фізіологічних сигналів пацієнта [15].

1.4.4 Організаційні та допоміжні системи

Такі системи вирішують немедичні проблеми а їх розвиток спрямований на підвищення ефективності роботи медичного персоналу, покращення ведення медичної документації та оптимізацію клінічних процесів. Вони дозволяють забезпечити інтеграцію різних етапів лікування в єдине інформаційне середовище, що не тільки покращує досвід медичного персоналу, а і дозволяє інсування організованого зберігання даних пацієнта.

Важливим технічним аспектом функціонування таких систем є забезпечення здатності різних інформаційних систем і медичних пристроїв до обміну та коректної інтерпретації даних. Тобто створення систем, які дозволяють інтегрувати інформацію з моніторів пацієнта, анестезіологічних станцій, лабораторних систем та електронних медичних записів у єдину цифрову екосистему. Створення такої системи дозволяє використання ШІ.

Інтеграція штучного інтелекту в систему керування електронними медичними записами дозволяє автоматизувати документування перебігу анестезії, забезпечувати безперервне оновлення даних пацієнта в режимі реального часу та об’єднувати інформацію з різних джерел моніторингу, підтримувати рішення за стилем анестезіолога і проводити аналітику даних [14]. У результаті формується клінічна картина пацієнта, яка може використовуватися як для оперативного прийняття рішень, так і для подальшого аналізу та оптимізації медичних процесів. Такий підхід зменшує навантаження на медичний персонал і підвищує точність та повноту медичної документації.

Окремим напрямом є використання ШІ для підтримки організації післяопераційного маршруту пацієнта. Системи можуть допомагати у прогнозуванні потреби в інтенсивній терапії та ймовірності переведення до відділення інтенсивної терапії після операції. Це сприяє більш ефективному плануванню ресурсів, ліжко-місць та персоналу, а також дозволяє завчасно готуватися до можливих ускладнень.

Розвиток інтелектуальних систем у цій сфері сприяє стандартизації клінічних процесів та покращенню координації між різними етапами надання медичної допомоги. Це робить систему анестезіологічного супроводу більш узгодженою, ефективною та орієнтованою на пацієнта.

1.5 Огляд наукових досліджень у сфері інтелектуального прогнозування анестезії

1.5.1 Дані та вхідні сигнали

Багато сучасних анестезіологічних досліджень базуються на відкритій базі даних VitalDB (що містить високочастотні записи понад 6388 операційних

випадків), а також на записах великих закордонних клінічних центрів отриманих за домовленістю (зокрема, лікарні Сеульського національного університету, Західнокитайської лікарні, університетської лікарні Окаями та університету Хенань [22-28]. Використання масивів даних, зібраних з реальних пацієнтів дозволяє моделям враховувати індивідуальні особливості пацієнтів та варіативність медичних протоколів.

Вхідні сигнали для побудови моделей прогнозування та керування дозуванням формують простір ознак, який умовно поділяється на кілька функціональних категорій.

Першу категорію становлять фармакологічні часові ряди, що описують динаміку введення анестетиків і анальгетиків. Найчастіше розглядається синергічна дія пропофолу та реміфентанілу. Дані про інфузію включають швидкість введення, кумулятивні дози, болюсні введення та оцінки концентрацій у ефект-сайті (Ce) і плазмі (Cp) [22].

Друга категорія охоплює незімінні в моменті характеристики пацієнтів. До них належать демографічні показники і параметри тіла : вік, стать, маса тіла, зріст, індекс маси тіла, класифікація стану пацієнта за ASA. Ці параметри використовуються, оскільки вони визначають індивідуальні особливості метаболізму, об’єм розподілу препаратів та чутливість центральної нервової системи до анестетиків [26].

Третю групу формують життєві показники та гемодинамічні часові ряди. Сюди входять артеріальний тиск, частота серцевих скорочень, сатурація кисню, а також концентрація вуглекислого газу на кінці видиху. На відміну від статичних показників ці параметри відображають активну реакцію нервової системи пацієнта на хірургічне втручання та фармакологічний вплив, і виступають маркерами глибини анестезії та фізіологічної стабільності пацієнта [25]. Це робить їх критично необхідними для при визначенні дозування препаратів.

Четверта категорія представлена індексними показниками глибини анестезії. Самим відомих з них є біспектральний індекс, який отримується шляхом спектрального аналізу ЕЕГ-сигналів, і часто застосовується в анестезіології. Альтернативно застосовуються індекс стану пацієнта (PSI) [27] та індекс Narcotrend [26]. Використання різних індексів дозволяє підвищити стійкість моделей до варіацій у способах оцінки глибини седації.

1.5.2 Архітектури та навчання моделей

Зазвичай на практиці для визначення рівня введеня препаратів використовують статистичні та фармакокінетичні моделі. Але сучасні публікації демонструють поступовий перехід до нейромережевих архітектур. Якщо ранні системи автоматизованого дозування здебільшого базувались на PID-регуляторах, то сучасні дослідження використовують методи глибокого навчання.

Найбільш поширеним підходом у роботах останніх років є використання рекурентних нейронних мереж LSTM та GRU. Це пояснюється тим, що процес дії анестетиків має виражену часову залежність: зміна швидкості інфузії впливає на фізіологічний стан пацієнта не миттєво, а через певний часовий інтервал. LSTM-моделі використовуються для прогнозування змін BIS [22][26] та оцінки майбутньої потреби у реміфентанілі [23][25]. Рекурентні мережі завжди забезпечують кращу стабільність прогнозування порівняно з традиційними регресійними методами, але найкраще вони себе показують при обробці даних на тривалих операціях.

Згорткові нейронні мережі застосовуються для виділення ознак і зменшення розмірів часових рядів. Так є робіти, де хгорткові шари використовуються для автоматичного виділення локальних патернів у життєвих

сигналах, після чого рекурентні модулі аналізують часову еволюцію цих ознак [24]. Такий підхід дозволяє скоротити розмір даних при цьому не змінивши рівень інформативності про довготривалі фармакологічні ефекти. Гібридна архітектура CNN-LSTM-ANN [23] змогла досягти меншої похибки прогнозування дози порівняно з простими LSTM-архітектурами.

Трансформер-архітектури [22] здатні краще аналізувати довгострокові залежності між фармокологічними і фізичними показниками пацієнта. На відміну від рекурентних мереж, трансформери використовують механізм уваги, який дозволяє моделі визначати, які саме часові фрагменти сигналу мають найбільший вплив на поточний прогноз.

Але поки що базовим класом моделей для досліджень залишаються рекурентні нейронні мережі, зокрема LSTM та GRU. Їх ключова перевага полягає у здатності зберігати довгострокову пам’ять про попередні стани системи і утримувати стан пацієнта, що є критично важливим для фармакокінетичних процесів, де ефект введеного препарату проявляється із затримкою та накопиченням. При цьому трансформер-архітектури явно мають значно кращі перспективи завдяки високій інтерпретованості.

Нейро-нечіткі системи, зокрема ANFIS [28], поєднують нейронні мережі з нечіткою логікою. Такі моделі імітують клінічне мислення лікаря, використовуючи інтерпретовані правила типу «якщо–то», що робить їх більш зрозумілими для медичного персоналу. Такі моделі здатні визначати рівень ін’єкції препарату без використання біспектрального індексу і, що більш важливо, моделі такого типу можуть бути інтерпретовані працівниками медичного закладу з мінімальними змінами.

Ще одним перспективним напрямом є застосування навчання з підкріпленням [27]. На відміну від класичних моделей прогнозування, RL-підхід розглядає систему дозування як задачу послідовного прийняття рішень. Агент отримує інформацію про поточний стан пацієнта та навчається змінювати швидкість інфузії таким чином, щоб утримувати цільовий рівень

седації. Архітектури типу актор-критик, їх особливість полягає у використанні показника переваги, який визначає, наскільки обрана дія краща за середньоочікувану для даного стану пацієнта. Це значно зменшує варіативність під час навчання та робить модель стабільнішою. Також моделі отримані навчанням без учителя дозволяють покращити рівень дози анастетика за можливості, так в [27] вдалося зменшити використання пропофолу на 14%.

Використовуються також різні методи оптимізації та адаптації моделей. У більшості робіт навчання штучних нейронних мереж здійснюється методом зворотного поширення помилки. Іноді застосовують альтернативні методи оптимізації, зокрема генетичні алгоритми, які дозволяють ефективно підбирати параметри системи керування та мінімізувати похибку дозування в умовах складної нелінійної динаміки [28]. Увага також приділяється адаптації моделей до міжпацієнтської варіабельності. Для цього використовується архітектура AdaRNN, здатна коригувати внутрішні представлення моделі відповідно до фізіологічних особливостей конкретного пацієнта [26]. Паралельно досліджуються методи дистиляції знань, у межах яких інформація від великих високоточних моделей переноситься до компактніших систем, придатних для використання безпосередньо на клінічному обладнанні з обмеженими обчислювальними ресурсами.

1.5.3 Метрики оцінки та результати

Основною ціллю моделей для прогнозування рівня дозування є не тільки досягнення метрик ефективності моделі зі сторони математичної точності, а і оцінка клічної доцільності застосування моделі у реальних умовах. Через це використовуються метрики, які дозволяють одночасно оцінити стабільність

підтримки анестезії, точність прогнозування та економічну ефективність використання препаратів.

Для задач безперервного прогнозування дозування або рівня глибини анестезії найчастіше застосовуються середньоквадратична помилка, середня абсолютна помилка та середня абсолютна відсоткова помилка. Так використання моделей заснованих на рекурентних мережах дозволило знизити середню абсолютну помилку до приблизно 12-16% [23].

Окрему увагу в літературі приділяють прогнозуванню моментів необхідності зміни швидкості інфузії. У таких роботах задача формулюється як проблема класифікації, а основною метрикою стає ROC-AUC. Зокрема, у роботах із прогнозування потреби у збільшенні дози реміфентанілу LSTM-моделі демонстрували значення ROC-AUC близько 0.75 [25].

У роботах, присвячених автоматизованому контролю інфузії, значна увага приділяється стабільності системи керування. RL-архітектура порівнюється з класичними PID-регулятором. В результаті системи навчання з підкріпленням демонструють меншу кількість нестабільностей цільової змінної рівня седації та краще утримують біспектральний індекс у заданому діапазоні. При цьому найкращий результат показала модель, націлена на заниження дози, таким чином цей підхід може зменшувати рівень використання анальгетику під час операції.

Також аналізується економічна ефективність запропонованих систем. Зокрема, моделі на основі ANFIS та генетичних алгоритмів дозволяли зменшити використання пропофолу приблизно на 14 % без погіршення стабільності анестезії [28]. На думку дослідників, це демонструє перспективність використання інтелектуальних систем не лише для підвищення точності дозування, але і для оптимізації витрат препаратів та зниження ризику побічних ефектів.

1.5.4 Обмеження та клінічна перспектива

Сучасні моделі все ще перебувають переважно на стадії експериментальних або пілотних клінічних досліджень. Основною причиною цього є складність перенесення результатів, отриманих у контрольованих умовах, до реальної клінічної практики.

Однією з найбільш обговорюваних проблем є відсутність єдності моделей. Багато систем навчаються на даних окремих лікарень, що призводить до адаптації до конкретних протоколів анестезії, типів обладнання та локальних особливостей моніторингу. При спробі використання моделей поза навчальним середовищем, створених тренувальним набором даних, її ефективність передбачення значно зменшується. Ця проблема погіршується для систем, які навчаються зучитилем, оскільки в такиму випадку дані упереджені персональним впливом на рівень введення препаратів лікаря-анестезіолога.

Іншим суттєвим обмеженням є проблема інтерпретованості рішень. Складні нейромережеві архітектури, функціонують як чорні скриньки, через що лікарю складно оцінити причини зміни дози, запропонованої системою. Саме тому у сучасних роботах дедалі частіше застосовуються методи для пояснення ШІ, зокрема SHAP [25] та LIME, які дозволяють оцінити внесок окремих параметрів у кінцевий прогноз моделі.

Значну увагу дослідники приділяють проблемі якості вхідних даних. У реальних умовах операційної ЕЕГ-сигнали, пульс, сатурація і всі інші показники гемодинаміки містять шуми, артефакти руху або пропуски вимірювань [26]. Це створює ризик формування помилкових рекомендацій щодо дозування. У зв'язку з цим активно розроблюються способи фільтрації сигналів, відновлення відсутніх даних і збільшення сталості моделей до викидів.

Не зважаючи існуючі проблеми подальша розробка моделей є перспективним напрямком для дослідження. Зараз моделі мають виконувати консультативну функцію [27], допомагаючи анестезіологу оцінювати тенденції зміни стану пацієнта та прогнозувати потребу в корекції дозування, а не повністю замінювати клінічне рішення лікаря.

1.5.5 Можливості майбутніх досліджень

Аналіз сучасної літератури показує, що подальший розвиток використання ШІ для анестезіологічного контролю має декілька напрямків, основним з яких є збільшення рівня персоналізації. Це дозволить краще використовувати і інтерпретувати інформацію про індивідуальні особливості пацієнта.

Значний інтерес викликають багатомодальні підходи, які поєднують дані інфузії, життєві показники, електроенцефалографічні сигнали та інформацію з електронних медичних карток. Інтеграція даних про супутні захворювання, попередню медикаментозну терапію та історію оперативних втручань може суттєво підвищити точність персоналізованого прогнозування.

Використання більш новітніх моделей і їх варіацій може значно покращити точність роботи систем. Більшість робіт використовують рекурентні нейронні мережі, які були стандартним підходом для обробки часових рядів, але все більше поступаються в ефективності іншим архітектурам.

Паралельно активно досліджується проблема оптимізації моделей для роботи в режимі реального часу. Це є необхідною умовою інтеграції систем штучного інтелекту безпосередньо в інфузійні та моніторингові системи операційних.

Окремий напрям майбутніх досліджень пов’язаний із розвитком інтерпретованого штучного інтелекту. Більшість авторів наголошують, що без пояснюваності рішень навіть високоточні системи матимуть обмежене клінічне застосування. Це викликає неохідність в створенні моделей, які здатні аргументовано пояснити причини зміни дози препарату і вказати, які показники мали найбільший вплив на прийняте системою рішення.

Іншим, доволі перспективним напрямом досліджя може бути створення повноцінної системи автоматичного замкненого керування. Така система буде здатна підтримувати цільовий рівень анестезії без постійного втручання лікаря. Однак для створення і використання такої системи необхідно розробити усі елементи згадані в цьому підпункті.

Висновки до розділу 1

У першому розділі було проведено аналіз предметної області автоматизованого підбору препаратів для анестезії та розглянуто клінічні, фармакологічні й технологічні аспекти сучасної анестезіології. Обгрунтовано актуальність дослідження задачі автоматизованого підбору препаратів для анестезії.

Розглянуто теоретичні засади анестезіології як дисципліни, що охоплює передопераційний, інтраопераційний та постопераційний етапи і основні препарати, які застосовуються для проведення анестезії в інтраопераційному періоді.

Досліджено сфери використання штучного інтелекту в анестезіології. Основними з них є предопераційна оцінка параметрів пацієнта, оцінка шансу ускладнень під час або після операції, системи безперервного моніторингу стану і системи підтримки інтраопераційних рішень.

Досліджено роботи, присвячені вирішенню задачі прогнозування рівня та параметрів ін’єкції анестетичних препаратів під час проведення загальної анестезії. У цих дослідженнях розглядаються підходи до автоматизованого або напівавтоматизованого керування дозуванням лікарських засобів на основі аналізу високочастотних фізіологічних сигналів та історії введення препаратів. Основна увага приділяється побудові різноманітних моделей, здатних відтворювати складну нелінійну залежність між швидкістю інфузії, індивідуальними характеристиками пацієнта та динамікою його фізіологічного стану під час операційного втручання. Основною моделлю для аналізу часових даних є рекурентні нейронні мережі архітектури LSTM.

РОЗДІЛ 2 НЕЙРОМЕРЕЖЕВІ МЕТОДИ ПРОГНОЗУВАННЯ ДОЗ АНЕСТЕЗІОЛОГІЧНИХ ПРЕПАРАТІВ

2.1 Загальні відомості про штучні нейронні мережі

Штучні нейронні мережі (ШНМ) – це обчислювальні моделі, натхненні біологічними нейронними мережами, такими як людський мозок. Вони мають довгу історію. Вона починається з створення перцептрону Розенблата, який вирішує задачу лінійного розділення простору. Перцептрон оперує вхідним вектором ознак, для якого кожен компонент відповідає окремій характеристиці вхідних даних, та обчислює зважену лінійну комбінацію з подальшим застосуванням функції активації, що формалізується формулами, наведеними далі.

$$X = (x_1, x_2, x_3.., x_n),$$
$$z = \sum_{i=1}^{n} w_i x_i + b = w^T x + b.$$
$$y = \sigma(z).$$

де x_i – значення ознак вхідних даних; w_i – вага ознаки; b – зсув, σ – функція активації.

Подальший розвиток теорії штучних нейронних мереж передбачає розгляд їх ключових складових, які визначають як архітектуру моделей, так і особливості їх навчання та функціонування.

2.1.1 Функції активації

Функції активації є ключовим елементом штучних нейронних мереж, оскільки вони вводять нелінійність у модель. Саме завдяки нелінійним перетворенням багатошарові мережі здатні апроксимувати складні залежності між вхідними та вихідними даними. Без функцій активації композиція декількох шарів зводилася б до еквіваленту однієї лінійної трансформації. Важливою властивістю будь-якої функції активації є наявність простої для обчислення першої похідної, що суттєво спрощує процес навчання нейронної мережі.

Однією з класичних функцій активації є сигмоїдна функція наведена в формулі 2.1.

$$\sigma(z) = \frac{1}{1+e^{z}}. \tag{2.1}$$

Вона відображає значення в інтервал (0,1), що дозволяє інтерпретувати вихід як ймовірність. Сигмоїда широко використовувалась у ранніх нейронних мережах і іноді застосовується для задач класифікації для інтерпретації результату мережі, через вищеописану властивість. Її недоліком є насичення на великих за модулем значеннях z, що призводить до проблеми зникнення градієнтів під час навчання.

Іншою функцією активації є гіперболічний тангенс:

$$\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}.$$

На відміну від сигмоїди, він відображає значення в діапазон (−1,1) і є центрованим відносно нуля.

Найбільш поширеною функцією активації в сучасних глибоких мережах є Rectified Linear Unit (ReLU) наведена в формулі.

$$\sigma(z) = \begin{cases} z, & z \geq 0 \\ 0, & z < 0 \end{cases}.$$

ReLU є обчислювано простою та зменшує ефект зникнення градієнтів для додатних значень. Завдяки цьому вона значно покращує швидкість і стабільність навчання глибоких мереж. Її недоліком є відсутність навчання нейронів з негативними значеннями.

Модифікацією функції ReLU, яка частково усуває зазначену проблему, є Leaky ReLU, що задається наступним чином:

$$\sigma(z) = \begin{cases} z, & z \geq 0 \\ az, & z < 0 \end{cases},$$

де $a \ll 1$.

Завдяки наявності a Leaky ReLU дозволяє зменшити проблему зникнення градієнта та покращує стабільність навчання глибоких нейронних мереж.

Для задач багатокласової класифікації часто використовується softmax-функція, яка перетворює вектор значень у розподіл ймовірностей:

$$softmax(z_i) = e^{z_i} / \sum_j e^{z_j}.$$

Softmax забезпечує інтерпретацію виходу моделі як ймовірнісного розподілу між класами.

2.1.2 Багатошаровий перцептрон (MLP)

Перцептрон здатний реалізовувати лише лінійні розділення. Для моделювання складніших залежностей використовують багатошарові нейронні мережі. Наприклад, для мережі з одним прихованим шаром обчислення мають такий вигляд.

$$h = \sigma\left(W^{(1)}x + b^{(1)}\right),$$
$$y = \sigma\left(W^{(2)}h + b^{(2)}\right),$$

де y – вихід мережі; h – вектор значень прихованого шару; $W^{(l)}$ та $b^{(l)}$ – матриці ваг та вектори зміщень відповідних шарів; σ – функція активації.

Кожен шар мережі виконує нелінійне перетворення виходу попереднього шару. Таким чином, MLP реалізує композицію функцій, у якій кожен наступний рівень формує більш абстрактне та узагальнене представлення вхідних даних. Подібна структура дозволяє поступово виділяти значущі ознаки, переходячи від простих локальних залежностей до комплексних властивостей існуючих в вхідному просторі.

Згідно з теоремою про універсальну апроксимацію [29], нейронна мережа з одним прихованим шаром і достатньою кількістю нейронів здатна апроксимувати довільну неперервну функцію:

$$f(x) \approx \sum_{j=1}^{m} a_j \sigma\left(w_j^T x + b_j\right),$$

У класичному доведенні теореми універсальної апроксимації використовувалась сигмоїдальна функція активації, представлена у формулі (2.1). Цей результат демонструє здатність відносно простої архітектури до відтворення складних функціональних залежностей.

Однак на практиці для досягнення високої точності така мережа може потребувати надзвичайно великої кількості нейронів. Тому замість одного дуже широкого шару зазвичай використовують глибокі багатошарові мережі. Це дозволяє представляти складні залежності як послідовність простіших перетворень, що виконуються на різних рівнях моделі.

2.1.3 Функції втрат

Функції втрат є ключовим елементом процесу навчання нейронних мереж, оскільки вони визначають міру невідповідності між прогнозованими значеннями моделі та реальними даними. Основною метою навчання є мінімізація значення функції втрат шляхом підбору оптимальних параметрів моделі.

Вибір функції втрат залежить від типу задачі. Для задач регресії, де необхідно прогнозувати неперервні значення, найчастіше використовується середньоквадратична похибка:

$$L_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2,$$

де $\widehat{y_i}$ – прогнозоване значення моделі; y_i – справжнє значення; n – розмір вибірки.

MSE штрафує за великі відхилення сильніше завдяки зведенню у квадрат, що робить її чутливою до викидів.

Альтернативою є середньоабсолютна похибка:

$$L_{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y_i}|.$$

MAE є більш стійкою до викидів, оскільки не зводить похибки у квадрат, однак має недолік – вона не є диференційованою у точці нуля, що ускладнює застосування градієнтних методів оптимізації.

Для задач бінарної класифікації найпоширенішою функцією втрат є бінарна крос-ентропія, що базується на принципі максимізації правдоподібності:

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)),$$

де $\hat{y}_i$ – прогнозована ймовірність приналежності до класу 1; $y_i \in \{0,1\}$ – справжня мітка класу; n – розмір вибірки.

Окрім стандартних функцій втрат, у практиці глибокого навчання застосовуються й спеціалізовані варіанти.

У задачах із суттєвим дисбалансом класів стандартна крос-ентропія виявляється неефективною: модель швидко навчається передбачати домінуючий клас та ігнорує рідкісні події. Для вирішення цієї проблеми була запропонована Focal Loss [30]:

$$L_{Focal} = -\frac{1}{n}\sum_{i=1}^{n}(y_i(1 - \hat{y}_i)^{\gamma}\log(\hat{y}_i) + (1 - y_i)\hat{y}_i^{\gamma}\log(1 - \hat{y}_i)),$$

де γ – параметр фокусування.

При $\gamma = 0$ вираз вироджується у стандартну бінарну крос-ентропію. А для $\gamma > 0$ зменшується внесок класів тих, які модель вже класифікує правильно з високою впевненістю. Це змушує навчання зосереджуватись на складних і рідкісних випадках.

Функція втрат Хьюбера [31] є гібридом між MSE та MAE і усуває їхні головні недоліки: надмірну чутливість до викидів у MSE та відсутність гладкості в MAE:

$$L_{Huber} = f(x) = \begin{cases} \frac{1}{2\delta}(y_i - \widehat{y_i})^2 + \frac{1}{2\delta}, & |y_i - \widehat{y_i}| \leq \delta \\ |y_i - \widehat{y_i}| & , \ |y_i - \widehat{y_i}| > \delta \end{cases},$$

де δ – поріг, що визначає межу між квадратичною та лінійною областями.

При малих відхиленнях функція втрат Хьюбера поводиться як MSE, забезпечуючи гладкий градієнт і швидку збіжність поблизу мінімуму. При великих відхиленнях перехід до лінійного штрафу запобігає домінуванню окремих аномальних зразків у градієнті.

У сучасних системах глибокого навчання нерідко виникає потреба одночасно вирішувати кілька пов'язаних задач у рамках однієї моделі. У такому разі загальна функція втрат формується як зважена сума часткових функцій втрат кожної підзадачі:

$$L_{Total} = \sum_{k=1}^{K} w_k * L_k,$$

де K – кількість задач; L_k – функція втрат k–ї підзадачі; а w_k – ваговий коефіцієнт, що визначає відносну значущість кожної задачі в процесі навчання.

Коректний підбір вагових коефіцієнтів є нетривіальною задачею: надмірне збільшення ваги однієї підзадачі може призвести до деградації якості навчання решти компонент, тоді як рівномірний розподіл ваг не завжди відображає реальну пріоритетність підзадач у контексті конкретної прикладної системи.

2.1.4 Навчання штучної нейронних мереж

Навчання нейронних мереж проходить на ідеї мінімізації функції втрат шляхом коригування параметрів моделі, до яких належать ваги та зсуви.

Стандартною способом досягнення цього є обчислення похідні функції втрат від кожного параметра моделі. Це реалізується за допомогою методу зворотного поширення помилки.

Метод зворотного поширення помилки використовує правила ланцюга, що дозволяє ефективно обчислювати градієнти складної композиції функцій. Ідея полягає в послідовному обчисленні похідних від виходу мережі до її вхідних шарів, що дає змогу визначити, як кожен параметр впливає на значення функції втрат.

Ефективність методу зворотного поширення помилки полягає в тому, що обчислення градієнтів для всіх параметрів виконується за час, пропорційний одному прямому проходу через мережу. Це робить можливим навчання великих багатошарових моделей на великих обсягах даних.

Після обчислення градієнтів параметри оновлюються за допомогою методів оптимізації. Найпростішим із них є градієнтний спуск , який виконує оновлення параметрів у напрямку протилежному градієнту функції втрат:

$$\theta' = \theta - \alpha \nabla_{\theta} \mathrm{L},$$

де θ – навчувані параметри моделі; α – швидкість навчання.

Як правило, навчання нейронних мереж не виконується для кожного окремого прикладу, а здійснюється на основі групи спостережень (batch). У такому випадку застосовується стохастичний градієнтний спуск, який оновлює параметри моделі на основі обмеженої підвибірки навчальних даних.

Основна ідея стохастичного градієнтного спуску полягає в тому, що замість обчислення градієнта функції втрат на всьому наборі даних, він оцінюється за випадково вибраною підмножиною прикладів або навіть одним прикладом. Це дозволяє суттєво зменшити обчислювальну складність одного кроку навчання та зробити процес оптимізації більш ефективним для великих наборів даних.

Сучасним стандартом оптимізації є алгоритм Adam (Adaptive Moment Estimation) [32], який поєднує ідеї інерції та адаптивного масштабування швидкості навчання для кожного параметра окремо. Adam забезпечує стабільну та швидку збіжність у більшості практичних задач і широко використовується як оптимізатор за замовчуванням у глибокому навчанні.

2.1.5 Регуляризація нейронних мереж

Регуляризація є важливим підходом у навчанні нейронних мереж, спрямованим на зменшення перенавчання підвищення здатності моделі до узагальнення. Перенавчання виникає тоді, коли модель занадто добре підлаштовується під навчальні дані, втрачаючи здатність коректно працювати з новими, раніше не баченими прикладами.

Одним із найпоширеніших методів регуляризації є L1 та L2 регуляризації, які передбачають додавання до функції втрат штрафного члена, що залежить від величини параметрів моделі. У випадку L2 регуляризації до функції втрат додається сума квадратів вагових коефіцієнтів.

$$L_{reg} = L + a \sum w_i^2.$$

Такий підхід сприяє зменшенню великих значень ваг і робить модель більш стабільною. L1 регуляризація, у свою чергу, базується на сумі абсолютних значень ваг і може призводити до розріджених рішень, тобто занулення частини параметрів.

Іншим популярним методом регуляризації є Dropout [33], який полягає у випадковому вимиканні частини нейронів під час навчання. Формально, кожен нейрон має ймовірність p бути неактивним на кожній ітерації навчання, що

змушує модель не покладатися на окремі ознаки або їх фіксовані комбінації. Це підвищує стійкість моделі до шуму та покращує її узагальнювальну здатність.

Метод Dropout інтерпретується як ефективна апроксимація навчання ансамблю підмереж із спільними параметрами. Під час навчання випадковим чином формується велика кількість різних архітектур, тоді як на етапі тестування використовується повна мережа з відповідним масштабуванням активацій, що забезпечує стабільність передбачень.

2.1.6 Нормалізація даних та ознак

Нормалізація даних є одним із ключових етапів попередньої обробки, що безпосередньо впливає на якість навчання моделей машинного та глибокого навчання. Її основна мета полягає у приведенні різнорідних за масштабом ознак до узгодженого числового простору, що забезпечує коректну роботу алгоритмів оптимізації та зменшує числову нестабільність під час обчислень градієнтів.

У задачах, де ознаки мають суттєво різні діапазони значень, відсутність нормалізації може призводити до домінування ознак із більшими числовими значеннями, що уповільнює збіжність градієнтних методів оптимізації.

Одним із найбільш поширених методів є стандартизація, яка перетворює вихідні значення ознаки таким чином, щоб їх розподіл мав нульове математичне сподівання та одиничну дисперсію:

$$x' = \frac{x-\mu}{\sigma},$$

де μ – середнє значення x; σ – стандартне відхилення x.

Такий підхід дозволяє зменшити залежність моделі від абсолютних значень ознак та вирівняти їх внесок у функцію втрат. Стандартизація особливо

ефективна у випадках, коли дані мають близький до нормального розподіл. Але стандартизація є чутливою до викидів.

Іншим поширеним підходом є лінійне масштабування ознак до фіксованого діапазону [0,1]:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}},$$

де x_{min} – мінімальне значення ознаки x; x_{max} – максимальне значення ознаки x.

Цей метод зберігає відносні пропорції між значеннями, але обмежує їх числовий діапазон, що особливо корисно для алгоритмів, чутливих до масштабу вхідних даних. З іншої сторони нормалізація є вразливою до появи нових викидів у тестових даних, оскільки вони можуть виходити за межі заданого діапазону.

У практичних задачах, де дані містять значну кількість викидів, застосовується робастна (Robust) нормалізація, яка використовує медіану та міжквартильний розмах:

$$x' = \frac{x - Q_2}{Q_3 - Q_1},$$

де Q_1, Q_2, Q_3 – перший квартиль, медіана та третій квартиль розподілу ознаки x.

Окрему категорію становлять методи внутрішньої нормалізації, які застосовуються безпосередньо всередині архітектури нейронної мережі. Найбільш відомим є Batch Normalization, який виконує стандартизацію активацій на рівні групи спостережень. Його основним обмеженням є залежність від розміру групи, що може знижувати ефективність у задачах із малими вибірками або послідовними даними. Альтернативним підходом є Layer Normalization, яка виконує нормалізацію по всіх ознаках окремого прикладу.

2.2 Рекурентні нейронні мережі

На відміну від класичних прямих нейронних мереж, які припускають незалежність вхідних спостережень, рекурентні нейронні мережі призначені для обробки послідовних даних, де між елементами існують часові або порядкові залежності. До таких даних належать текст, мовлення, часові ряди, біосигнали тощо. Ключовою особливістю РНМ є наявність внутрішнього стану, який дозволяє враховувати інформацію з попередніх кроків послідовності.

На кожному кроці t мережа отримує вхідний вектор x_t, оновлює прихований стан o_t та генерує вихід y_t:

$$h_t = \sigma(W_x x_t + W_h h_{t-1} + b),$$
$$o_t = \varphi\big(W_y h_t + b\big),$$

де o_t – прихований стан(пам’ять) на кроці t; W_x – матриця вагів для вхідного вектору; W_h – матриця вагів для внутрішнього стану; W_y – матриця вагів для вихідного вектору; b – зсув; φ, σ – функції активації.

Таким чином, поточний стан мережі залежить як від поточного входу, так і від попереднього стану, що забезпечує накопичення контекстної інформації.

Важливою властивістю рекурентних нейронних мереж є те, що матриці вагів W_x, W_h, W_y та зсув b залишаються незмінними на всіх кроках часу. Це означає, що одна й та сама сукупність перетворень застосовується до кожного елемента послідовності незалежно від його позиції. Завдяки цьому значно зменшується кількість параметрів моделі та забезпечується можливість працювати з послідовностями довільної довжини.

На рисунку 2.1 відображено РНМ у двох формах – згорнутій і розгорнутій. Як видно зсув і матриці вагів не залежать від номеру ітерації. Така організація означає, що мережа однаково реагує на подібні закономірності

незалежно від їхнього положення в послідовності, що дозволяє накопичувати контекстну інформацію. Водночас спільність параметрів призводить до того, що під час навчання враховується внесок кожного кроку, що може ускладнювати процес через поступове зменшення або різке зростання впливу похибки при проходженні через довгі послідовності.

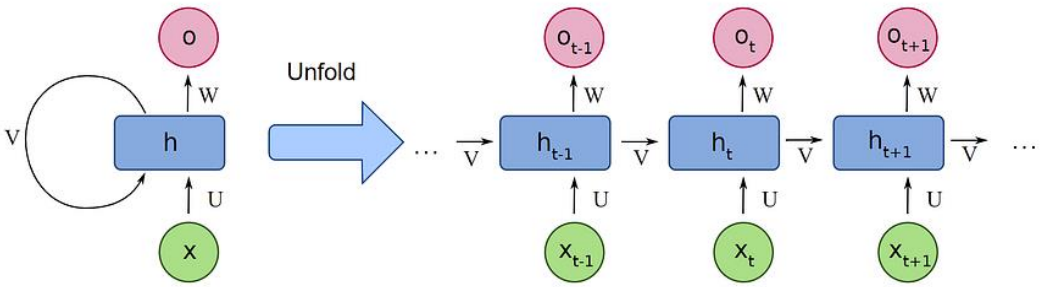


Рисунок 2.1 – Представлення RNN як у згорнутій, так і в розгорнутій формі[2]

2.2.1 Архітектури РНМ

Базовою формою є проста рекурентна мережа, яка реалізує принцип передачі прихованого стану між кроками послідовності згідно з формулами 2.2 і 2.3. Незважаючи на концептуальну простоту, така архітектура на практиці має суттєві обмеження, пов’язані з нестабільністю навчання. Зокрема, під час застосування методу зворотного поширення помилки в часі виникають проблеми зникаючого та вибухового градієнтів, через що знижується здатність моделі ефективно враховувати довготривалі залежності. Для часткового подолання цих обмежень були запропоновані розширені архітектури.

Одним із таких розширень є двонаправлені рекурентні нейронні мережі. У цій архітектурі обробка послідовності виконується одночасно у двох

[2]Джерело зображення: https://miro.medium.com/v2/resize:fit:1100/format:webp/1*SKGAqkVVzT6co-sZ29ze-g.png

напрямках: від початку до кінця та від кінця до початку. Такий підхід дозволяє моделі враховувати як попередній, так і наступний контекст для кожного елемента послідовності, що є корисним у задачах, де значення елемента залежить від повного контексту.

Архітектурно двонаправлені рекурентні нейронні мережі складається з двох незалежних рекурентних шарів. Прямий прихований стан обчислюється за формулою 2.2, тоді як зворотний прихований стан, замість h_{t-1} використовує h_{t+1}.

LSTM (Long Short-Term Memory). Архітектура LSTM, запропонована Хохрайтером та Шмідхубером [34], вирішує проблему зникаючого градієнта шляхом введення окремого вектора стану клітини c_t, який передається між кроками з мінімальними перетвореннями. Керування потоком інформації здійснюється трьома типами гейтів , що являють собою навчені сигмоїдні перетворення з виходами в діапазоні (0, 1).

Гейт забування f_t визначає, яка частина попереднього стану клітини має бути збережена:

$$f_t = \sigma\big(W_f x_t + U_f h_{t-1}\big).$$

Вхідний гейт i_t регулює, яка нова інформація буде записана до стану клітини:

$$i_t = \sigma(W_i x_t + U_i h_{t-1}).$$

Кандидат нового стану $\overline{c}_t$ формує потенційне оновлення пам’яті:

$$\overline{c}_t = \tanh(W_c x_t + U_c h_{t-1}).$$

Оновлення стану клітини поєднує збережену та нову інформацію:

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \overline{c_t}.$$

Вихідний гейт o_t та прихований стан h_t формуються як:

$$o_t = \sigma(W_x x_t + W_h h_{t-1} + b_0),$$
$$h_t = o_t \tanh(c_t).$$

Завдяки такій структурі LSTM здатна зберігати релевантну інформацію протягом сотень кроків послідовності, що робить її особливо ефективною для задач із довгостроковими залежностями.

GRU (Gated Recurrent Unit). GRU [35] є спрощеною альтернативою LSTM, яка зменшує складність архітектури за рахунок об'єднання частини механізмів керування пам'яттю. Зокрема, у GRU гейт забування та вхідний гейт об'єднані в єдиний гейт оновлення z_t, а для додаткового контролю впливу попереднього стану використовується гейт скидання r_t. Така структура дозволяє моделі ефективно балансувати між збереженням попередньої інформації та оновленням новими даними без явного розділення стану клітини та прихованого стану.

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z),$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r),$$
$$\widetilde{h_t} = \tanh(W_h x_t + U_h(r_t * h_{t-1}) + b_h),$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \widetilde{h_t}.$$

GRU має меншу кількість параметрів порівняно з LSTM і демонструє порівнянну ефективність на багатьох задачах, забезпечуючи при цьому швидше навчання. Вибір між LSTM та GRU здебільшого визначається емпірично залежно від природи даних та обчислювальних обмежень.

2.2.2 РНМ та часові ряди

Рекурентні нейронні мережі є інструментом створеним для моделювання часових рядів. Вони явно враховують залежності між послідовними спостереженнями. У задачах прогнозування модель навчається відображенню $(x_1, x_2, \dots, x_t) \to x_{t+1}$, тобто на основі попередніх значень ряду передбачає наступне. Внутрішній прихований стан h_t виступає узагальненим стисненим представленням усієї попередньої історії послідовності.

Завдяки цьому РНМ здатні виявляти різноманітні структури в часових рядах: короткострокові локальні коливання, довгострокові тренди, а також циклічні та сезонні закономірності. Архітектури LSTM та GRU особливо добре підходять для рядів із нерегулярними або довгостроковими залежностями, оскільки механізми гейтів дозволяють вибірково накопичувати та відкидати інформацію залежно від її релевантності.

У задачах медичного прогнозування, зокрема при моделюванні динаміки введення анестезіологічних препаратів, часові ряди характеризуються високим рівнем шуму, нерівномірною структурою та наявністю різких пікових змін. Рекурентні нейронні мережі дозволяють враховувати контекст попередніх станів пацієнта та динаміку змін параметрів у часі, щоб повернути інформативне представлення даних.

2.2.3 Ієрархічні мережі уваги і механізм агрегаційної уваги для RNN

Ієрархічні мережі уваги [36] було запропоновано як ієрархічну архітектуру нейронної мережі з механізмом уваги для задачі класифікації документів. ЇЇ основною ідеєю є врахування багаторівневої структури тексту,

яка природно складається зі слів, речень та документів. На кожному рівні цієї ієрархії застосовуються рекурентні нейронні мережі для побудови представлень, а механізм уваги використовується для виділення найбільш інформативних елементів.

На першому рівні модель формує представлення слів у межах речення за допомогою рекурентної нейронної мережі. Отримані приховані стани кожного слова відображають контекстну інформацію з урахуванням сусідніх елементів послідовності. Далі для кожного прихованого стану обчислюється оцінка важливості, яка визначає внесок відповідного слова у формування представлення речення.

Механізм уваги задається у два етапи [36]. На першому етапі прихований стан проєктується у простір ознак проміжного рівня:

$$u_t = \tanh(Wh_t + b).$$

Після цього обчислюється скалярна оцінка важливості через подібність до навчуваного контекстного вектора:

$$a_t = softmax(u_t^T u_w),$$

де u_w – вектор, що моделює важливість ознак.

Отримані ваги використовуються для зваженого усереднення прихованих станів.

Ключовою особливістю запропонованого підходу є те, що механізм уваги не залежить від зовнішнього декодера або окремої цільової послідовності. Натомість він використовується як механізм агрегування інформації, що дозволяє моделі навчатися визначати важливість елементів послідовності.

Якщо спростити оригінальну схему, вилучивши нелінійне перетворення та контекстний вектор:

$$a_t = softmax(w^T h_t),$$
$$y = \sum_t (a_t\, h_t).$$

Таким чином, модель реалізує механізм зваженого агрегування прихованих станів рекурентної мережі у фіксоване представлення, де ваги визначаються виключно внутрішніми ознаками кожного стану без використання зовнішнього контексту або додаткової послідовності запитів.

2.3 Трансформер і механізм уваги

Архітектура трансформера, запропонована в 2017 [37], стала революційним кроком у розвитку глибокого навчання. На відміну від рекурентних мереж, трансформер повністю відмовляється від послідовної обробки та рекурентних компонентів, будуючи свою роботу виключно на механізмах уваги. Це рішення дозволяє моделювати залежності між будь-якими двома елементами послідовності напряму, незалежно від відстані між ними.

Вона складається з блоків кодувальника та декодувальника, кожен з яких містить шар багатоголової самоуваги, шар прямого поширення, залишкові зв’язки та шар нормалізації, як показано на рисунку 2.2. Для збереження інформації про порядок елементів послідовності, яка втрачається при відмові від рекурентності, застосовується позиційне кодування, що додається до вхідних векторів.

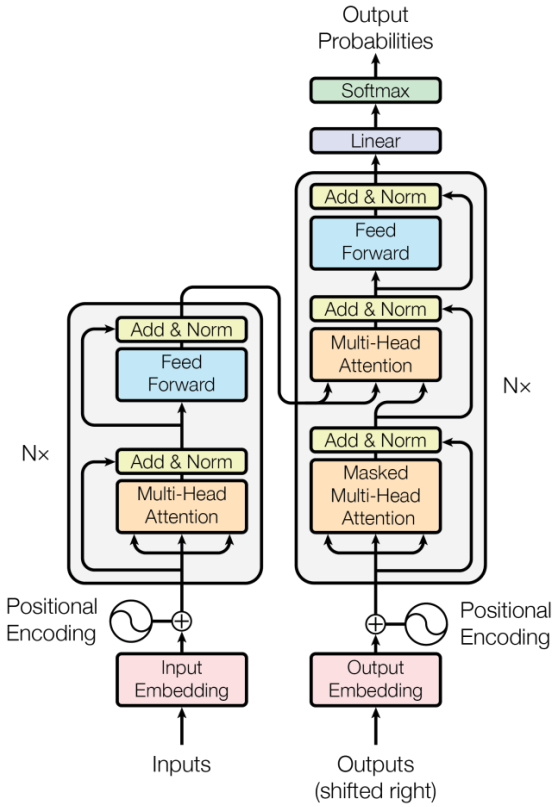


Рисунок 2.2 – Архітектура Трансформера [37]

2.3.1 Механізм самоуваги

Ключовим елементом трансформера є механізм самоуваги, що дозволяє кожному елементу послідовності зважено враховувати контекст усієї послідовності одночасно. На відміну від РНМ, де контекст передається через прихований стан і накопичується поступово, самоувага обчислює зв’язки між усіма парами позицій за один крок.

Для кожного елемента послідовності формується три вектори: запит Q, що описує, яку інформацію шукає елемент; ключ K , що описує, яку інформацію елемент пропонує; та значення V , що є фактичним вмістом, який передається. Якщо X – це матриця вхідних представлень, то:

$$\mathrm{Q} = XW_Q,$$
$$\mathrm{K} = XW_K,$$
$$\mathrm{V} = XW_V,$$

де W_Q, W_K, W_V – навчувані матриці вагів.

Самоувага обчислює зважену суму значень, де ваги визначаються схожістю між запитами та ключами:

$$Attention(Q,\ K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V.$$

Масштабування на $\sqrt{d_k}$ запобігає надмірно великим скалярним добуткам при високій розмірності, які б призводили до насичення функції softmax і зникнення градієнтів. Результуюча матриця уваги розміру $n * n$ явно кодує силу взаємодії між кожною парою позицій, що робить модель інтерпретованою.

Багатоголова увага. На практиці один механізм самоуваги може бути недостатнім для захоплення всіх типів залежностей у даних. Для вирішення цього використовується багатоголова увага, де h незалежних голів уваги паралельно обробляють послідовність, кожна зі своїм набором проекційних матриць $W_Q^{(i)}, W_K^{(i)}, W_V^{(i)}$. Результати всіх голів об’єднуються конкатенацією:

$$MultiHead(Q, K, V) = Concat\big(head_1, head_{2,\dots,} head_h\big),$$

де W_O – матриця вихідної проекції.

Кожна голова аналізує послідовність із власного підпростору зменшеної розмірності, завдяки чому різні голови можуть спеціалізуютися на різних аспектах залежностей.

2.3.2 Позиційне кодування

Оскільки механізм самоуваги є інваріантним до порядку елементів, то перестановка рядків не змінює структуру обчислень, лише переставляє відповідні рядки результату, тому трансформер потребує явного введення інформації про позиції. Для цього до вхідних векторів додають вектори позиційного кодування PE, що несуть інформацію про абсолютну або відносну позицію елемента в послідовності.

$$PE_{(pos,\,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{modle}}}}\right),$$

$$PE_{(pos,\,2i+1)} = cos\left(\frac{pos}{10000^{\frac{2i}{d_{modle}}}}\right).$$

Геометрично зростаючі частоти надають кожній позиції унікальність у просторі ознак, а синусоїдальна природа кодування дозволяє моделі узагальнюватись на послідовності довші за ті, що були в навчальній вибірці.

Адаптації для часових рядів. При застосуванні трансформерів до часових рядів синусоїдального кодування часто буває недостатньо, оскільки реальні часові дані мають специфічну структуру. У зв’язку з цим існують декілька варіантів позиційного кодування пристосованих для роботи з часовими рядами.

1. Навчуване позиційне кодування. Замість фіксованої формули кожній позиції і відповідає власний вектор, що оптимізується разом з іншими параметрами моделі. Це дозволяє гнучкіше адаптуватись до специфічних залежностей у даних, проте втрачається здатність до узагальнення на довші послідовності.
2. Відносне позиційне кодування [38]. Механізм уваги модифікується так, щоб враховувати відносну відстань $r = j - i$ між позиціями i та

j. Це досягається додаванням навчуваного зсуву до скалярного добутку $Q_i K_j^T$ залежно від значення r. Відносне кодування особливо важливе для часових рядів, де важлива не абсолютна позиція, а відносний часовий інтервал між спостереженнями.

2.3.3 Трансформер для табличних і медичних часових даних

У сучасних задачах машинного навчання архітектура трансформеру дедалі частіше адаптується для обробки нетекстових даних, зокрема табличних структур та медичних часових рядів. Це пов’язано з її здатністю моделювати складні нелінійні взаємозв’язки між ознаками одного рівня та ефективно працювати з послідовностями різної довжини.

Табличні дані зазвичай містять гетерогенні ознаки, які включають як числові, так і категоріальні змінні. У таких умовах класичні підходи, зокрема багатошарові перцептрони або методи градієнтного бустингу дерев рішень не завжди ефективно моделюють складні взаємозалежності між ознаками. Трансформерні архітектури пропонують альтернативний підхід, розглядаючи ознаки як послідовність токенів і застосовуючи механізм уваги для формування контекстно залежних представлень. Це дозволяє краще враховувати взаємозв’язки між ознаками, особливо у випадках високої розмірності простору або наявності прихованих залежностей. На практиці ефективність таких моделей найбільш проявляється на великих або складних за структурою наборах даних.

У задачах прогнозування часових рядів трансформери були адаптовані для ефективної роботи з довгими послідовностями та складними часовими залежностями. Одним із напрямів розвитку є оптимізація механізму уваги для зниження обчислювальної складності та підвищення здатності моделі до

узагальнення на довгих горизонтах прогнозу. У цьому контексті застосовуються спеціалізовані архітектури, серед яких Informer, що дозволяє ефективніше обробляти довгі часові ряди та виділяти глобальні закономірності в даних. Такі підходи є особливо корисними в задачах довгострокового прогнозування, де необхідно враховувати як короткострокові, так і віддалені залежності у часовій динаміці.

Окрему складність становлять медичні часові дані, зокрема електронні медичні записи, які характеризуються нерегулярністю спостережень, наявністю пропусків та різною частотою вимірювань. Це ускладнює застосування стандартних моделей послідовностей і вимагає спеціальних механізмів обробки часу та маскування даних. У таких умовах трансформерні підходи дозволяють будувати узагальнені представлення медичних процесів у часі, враховуючи нерівномірність спостережень і зберігаючи інформацію про динаміку стану пацієнта. Це робить їх перспективними для задач клінічного прогнозування та підтримки прийняття рішень у медицині.

2.4 Архітектура Informer

Informer є трансформерною архітектурою, спеціально адаптованою для задачі довгострокового прогнозування часових рядів [39]. Основна проблема застосування класичного Transformer у таких задачах полягає у надмірних витратах обчислювальних ресурсів та пам’яті через квадратичну складність механізму самоуваги. Крім того, при збільшенні довжини вхідних послідовностей різко зростає потреба у пам’яті при використанні багатошарової структури кодувальника.

Архітектура Informer зберігає загальну схему кодувальник–декодувальник, однак ключові зміни спрямовані саме на кодувальник, оскільки

він відповідає за формування стійкого прихованого представлення довгих послідовностей

Кодувальник Informer призначений для виділення довготривалих залежностей у часовому ряді та формування узагальненого представлення вхідних даних. На відміну від класичного Transformer, у якому всі шари кодувальника працюють із повною довжиною послідовності, Informer вводить механізми, які зменшують обсяг даних у проміжних шарах, що дозволяє значно знизити витрати пам’яті та прискорити навчання.

Для зменшення обчислювальної складності механізму самоуваги в архітектурі Informer було запропоновано модифікацію класичного механізму уваги, яка отримала назву ProbSparse self-attention.

Автори Informer виходять із припущення, що в механізмі самоуваги реальну роль відіграє лише мала кількість найбільш інформативних елементів. Для цього вводиться показник розподілу уваги для кожного запиту на основі KL-розбіжності:

$$M(q_i, K) = \ln \sum_{j=1}^{L_K} e^{\frac{q_i k_j^T}{\sqrt{d}}} - \frac{1}{L_K} \sum_{j=1}^{L_K} \frac{q_i k_j^T}{\sqrt{d}},$$

де q_i – і-й вектор запиту; K – множина ключів; k_j – j-й вектор ключа; L_K – кількість ключів; d – розмірність векторів ключі або запитів.

Якщо $M(q_i, K)$ є великим, то запит q_i формує більшу кількість розподілу ключів, що означає, він несе значущу інформацію для моделювання залежностей.

Щоб уникнути надмірних витрат, Informer застосовує наближену оцінку макси-середню, яка є значно швидшою у розрахунку:

$$M(q_i, K) = \max_j \left(\frac{q_i k_j^T}{\sqrt{d}}\right) - \frac{1}{L_K} \sum_{j=1}^{L_K} \frac{q_i k_j^T}{\sqrt{d}}.$$

На основі цієї міри Informer відбирає лише u найбільш значущих запитів, де:

$$u = c \ln(L_Q),$$

де c – емпірично отриманий коефіцієнт.

Це дає змогу суттєво зменшити обчислювальну складність механізму самоуваги, оскільки замість обробки всіх запитів на кожному рівні кодувальника модель виконує повний розрахунок самоуваги лише для обмеженої кількості найбільш інформативних запитів.

Окрім зменшення кількості обчислень самоуваги, Informer використовує механізм, спрямований на поступове скорочення довжини внутрішніх представлень у багатошаровому кодувальнику. Цей підхід названо дистиляцією самоуваги.

Ідея полягає в тому, що навіть після застосування розрідженої самоуваги проміжні карти ознак можуть містити надлишкову інформацію. Тому після кожного шару кодувальника застосовується згорткова обробка та операція максимального об’єднання, яка зменшує часову розмірність у два рази:

$$X_{j+1}^t = MaxPool(ELU(Conv1d([X_j^t]_{AB}))),$$

де $[X_j^t]_{AB}$ – вихід блоку уваги на попередньому шарі; *Conv1d* – одновимірна згортка(з ядром в 3); *ELU* – функція активації, *MaxPool* – максимізуючий агрегувальний шар (з розміром 2).

Таким чином кожен наступний рівень кодувальника працює з коротшою послідовністю, що суттєво знижує витрати пам’яті при збільшенні кількості шарів.

2.5 Метрики оцінки моделей

Вибір метрик оцінки є невід'ємною частиною процесу розробки та валідації моделей машинного навчання. Для задач регресії, де модель прогнозує неперервні значення, використовуються метрики, що кількісно характеризують відхилення між прогнозованими та реальними значеннями. Для задач класифікації, де модель передбачає належність зразка до одного з класів, застосовується окремий набір метрик, що оцінюють якість розмежування класів. У випадку багатозадачних моделей, які одночасно вирішують задачі різних типів, кожна підзадача оцінюється власним набором метрик відповідно до її природи.

2.5.1 Стандартні метрики для задачі регресії

Середня абсолютна похибка обчислюється як середнє абсолютних відхилень між прогнозованими $\widehat{y_i}$ та реальними y_i значеннями:

$$\mathrm{M}AE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y_i}|.$$

MAE має ту саму розмірність, що й цільова змінна, що робить її безпосередньо інтерпретованою в клінічних одиницях.

Середньоквадратична похибка (MSE) та її корінь (RMSE)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2,$$

$$RMSE = \sqrt{MSE}.$$

Квадратична природа MSE означає, що великі помилки штрафуються непропорційно сильно. У контексті дозування анестетиків це дає змогу накласти додатковий штраф при значному відхилені.

Коефіцієнт детермінації R^2відображає частку дисперсії цільової змінної, що пояснюється моделлю:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2},$$

де $\bar{y}$ – середнє значення по вибірці. $R^2 = 1$ відповідає ідеальному прогнозу.

2.5.2 Стандартні метрики для задачі бінарної класифікації

Для оцінки якості бінарних класифікаторів базовою концепцією є матриця невідповідностей, яка систематизує результати класифікації за чотирма категоріями: істинно позитивні результати (TP), істинно негативні (TN), хибно позитивні (FP) та хибно негативні (FN).

Найпростішою мірою є точність класифікації, що визначає частку правильно класифікованих зразків у загальній вибірці:

$$Acc = \frac{TP+TN}{TP+TN+FP+FN}.$$

Попри простоту, дана метрика є неінформативною в умовах дисбалансу класів, бо класифікатор, що відносить усі зразки до домінуючого класу, досягає формально високої точності без реальної розрізнювальної здатності.

Влучність визначає умовну ймовірність того, що зразок, класифікований як позитивний, є таким насправді:

$$Precision = \frac{TP}{TP+FP}.$$

Повнота характеризує частку правильно ідентифікованих позитивних зразків серед усіх реальних позитивних прикладів у вибірці:

$$Recall = \frac{TP}{TP+FN}.$$

Влучність та повнота перебувають у відношенні залежності: підвищення порогу прийняття рішення збільшує частку коректних позитивних передбачень, однак одночасно призводить до зростання кількості пропущених позитивних зразків, і навпаки. Цей компроміс унеможливлює одночасну максимізацію обох метрик і вимагає явного визначення пріоритетів залежно від прикладного контексту. Для єдиної агрегованої оцінки використовується F_1-score:

$$F1 = 2\frac{Precision*Recall}{Precision+Recall}.$$

Наведені вище метрики залежать від обраного порогу класифікації, що обмежує їх застосовність для порівняльного аналізу моделей. Для порогово-незалежної оцінки використовується ROC-крива, що описує залежність між часткою істинно позитивних результатів та часткою хибно позитивних при варіюванні порогу класифікації. Числовою характеристикою, яку можна отримати з неї є площа під ROC-кривою (AUC-ROC).

2.5.3 Клінічно-орієнтовані метрики

Стандартні метрики не завжди відображають клінічну значущість помилки. У анестезіології критичним є не лише середнє відхилення, а й характер розподілу помилок.

Систематичне зміщення (bias) визначається як середня похибка:

$$Bias = \frac{1}{n}\sum_{i=1}^{n}(\widehat{y_i} - y_i).$$

Позитивне зміщення означає систематичне завищення дози, а негативне – заниження. З клінічної точки зору ці два типи помилок мають різні наслідки: передозування анестетика загрожує пригніченням серцево-судинної або дихальної функції, тоді як недостатня доза може призвести до інтраопераційного пробудження пацієнта. Тому важливо знати значення систематичного зміщення моделі.

Для повнішої характеристики розподілу помилок доцільно також аналізувати найгірші прогнози, щоб оцінити поведінку моделі в найгірших сценаріях, які, незважаючи на малу частоту, є найбільш критичними в медичному контексті.

2.6 Постановка задачі

У межах цієї роботи ставиться задача побудови інтелектуальної системи, яка на основі комплексу вхідних клінічних даних пацієнта здійснює визначення оптимальних препаратів для проведення анестезії під час хірургічної операції. Система повинна враховувати як статичні характеристики пацієнта, що є відомими до початку операції, так і динамічні інтраопераційні показники, що змінюються в реальному часі протягом хірургічного втручання.

Формально задачу можна описати наступним чином. Нехай для кожного пацієнта визначено простір вхідних ознак, який складається з трьох основних груп даних.

Простір вхідних ознак складається з трьох груп даних.

До першої групи належать персональні характеристики пацієнта – статичні показники, що описують загальний стан організму до початку. Ці дані формують статичний вектор ознак операції $x^{(c)}$, який є повністю спостережуваним до початку оперативного втручання і залишається незмінним протягом усієї операції.

$$x^{(c)} = (age, sex, bmi, weight, asa, em, type, dep, cormack, htn, dm),$$

де age – вік пацієнта, $sex \in \{0,1\}$; bmi – індекс маси тіла; $weight$ – вага пацієнта; $asa \in \{1,2,3,4,5\}$ – класифікація пацієнта за системою ASA; em – бінарний показник, що вказує операція планова чи термінова; $type$ – тип операції; dep – відділення в якому проходить операція; $cormack$ – класифікація складності інтубації пацієнта; htn – бінарний показник наявності артеріальної гіпертензії; dm – бінарний показник наявності діабету.

До другої групи належать інтраопераційні показники – динамічні дані, що реєструються під час операції з певною частотою та утворюють часові ряди. Ця група включає: тривалість операції на кожен момент часу, значення біспектрального індексу, динаміку артеріального тиску, а також інформацію про препарати, що вводилися пацієнту під час операції.

Визначається матриця значень вектору часових рядів $X^{(t)} \in R^{T*6}$, де T - кількість часових кроків спостереження.

$$X^{(t)} = [s_1, s_2, \dots, s_T],$$

де кожен вектор стану s_t у момент часу t задається як:

$$s_t = (ART\_MBP_t, PLETH\_SPO2_t, BIS_t, ETCO2_t, PPF_t, RFTN_t),$$

де перші чотири компоненти є показниками приборів артеріальний тиск (мм рт. ст.), насичення крові киснем (%), глибина анестезії за біспектральним індексом

і концентрація вуглекислого газу при видоху (мм рт. ст.) відповідно. Останні дві компоненти є дозами препаратів пропофолу і реміфентанілу.

До третьої групи належать лабораторні показники пацієнта, отримані до або під час операції.

$$x^{(l)} = (cr, bun, gfr, aptt, fib, ptinr, hb, hct, alb, gluc),$$

де cr – рівень креатиніну в сироватці крові, що характеризує функцію нирок; bun – рівень азоту сечовини крові; gfr – швидкість клубочкової фільтрації; aptt – активований частковий тромбопластиновий час; fib – концентрація фібриногену; ptinr – міжнародне нормалізоване відношення протромбінового часу; hb – рівень гемоглобіну; hct – гематокрит; alb – рівень альбуміну; gluc – рівень глюкози в крові.

Вектор $x^{(l)}$ охоплює показники функції нирок, коагуляції, гематологічні параметри та метаболічні маркери. Діапазони параметрів наведено у відповідних нормативних одиницях вимірювання.

Вихідними даними системи є вектор прогнозованих швидкостей введення анестетиків у момент часу t:

$$y_t = (PPF_t, RFTN_t).$$

Таким чином, задача є регресійною: система передбачає неперервні значення доз двох препаратів, що відповідають оптимальній анестезії для конкретного клінічного профілю пацієнта.

Повний вхідний профіль пацієнта визначається як:

$$P = (x^{(c)}, X_{1:t-1}^{(t)}, x^{(l)}),$$

де $X_{1:t-1}^{(t)}$ – префікс часового ряду до поточного моменту.

Задача полягає у побудові відображення, що мінімізує функцію втрат на навчальній вибірці $\{(P_i, y_{t,i})\}$.

Ключовими вимогами до розроблюваної інтелектуальної системи є: висока точність класифікації з мінімізацією хибнонегативних помилок, оскільки у медичному контексті пропуск критично небезпечної комбінації має значно серйозніші наслідки, ніж надмірна обережність, здатність до обробки часових рядів змінної довжини, оскільки тривалість операцій може варіюватися, стійкість до пропущених значень у вхідних даних, оскільки частина даних може не збиратися.

Висновки до розділу 2

У другому розділі було розглянуто теоретичні основи побудови сучасних методів машинного навчання, що використовуються для аналізу складних багатовимірних даних у задачах прогнозування та прийняття рішень. Проаналізовано ключові елементи штучних нейронних мереж, зокрема функції активації, функції втрат, підходи до навчання, регуляризації та нормалізації, які забезпечують коректне налаштування моделей і стабільність процесу оптимізації.

Розглянуто рекурентні нейронні мережі як клас моделей, орієнтованих на обробку послідовних залежностей. Висвітлено їх основні архітектури та особливості застосування в задачах, де важливим є врахування історії спостережень та накопиченого стану системи.

Також проаналізовано архітектуру Transformer та механізм самоуваги як сучасний інструмент моделювання взаємозв’язків між ознаками. Розглянуто принципи позиційного кодування та застосування трансформерних підходів для

структурованих та медичних даних, що дозволяє ефективно виявляти складні залежності між змінними.

Окрему увагу приділено архітектурі Informer як оптимізованому варіанту трансформера, орієнтованому на підвищення ефективності обчислень та роботи зі складними послідовностями. Розглянуто основні підходи до оцінки якості моделей, включаючи регресійні, класифікаційні та клінічно орієнтовані метрики, що дозволяє комплексно оцінювати результати моделювання.

РОЗДІЛ 3 ПОБУДОВА ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ВИЗНАЧЕННЯ ПРЕПАРАТІВ ДЛЯ АНЕСТЕЗІЇ ПРИ ВИКОНАННІ ОПЕРАЦІЇ

3.1 Набір даних

Для даної роботи мною було обрано VitalDB[40]. Це відкритий набір даних, створений спеціально для сприяння дослідженням машинного навчання, пов’язаним з моніторингом життєво важливих показників у хірургічних пацієнтів. Цей набір даних містить багатопараметричні дані високої роздільної здатності з 6388 випадків, включаючи 486451 сигнальних та числових записів, 196 параметрів інтраопераційного моніторингу, 73 періопераційні клінічні параметри та 34 параметри часових рядів лабораторних результатів. Всі дані зберігаються у публічній хмарі і до набору даних можна вільно отримати доступ за допомогою інтерфейсів прикладного програмування та однойменної бібліотеки Python.

З точки зору клінічного охоплення, набір даних було отримано від пацієнтів, які перенесли некардіологічні (загальні, торакальні, урологічні та гінекологічні) планові або екстрені операції у 10 з 31 операційних лікарень Сеульського національного університету, Сеул, Республіка Корея.

Демогафічну характеристику бази даних VitalDB наведено в таблиці 3.1. Оскільки дані формуються шляхом інтеграції сигналів із різних типів медичного обладнання (монітори пацієнта, інфузійні помпи, апарати штучної вентиляції легень тощо), окремі клінічні випадки відрізняються як за складом доступних параметрів, так і за їх кількістю. Це означає, що для різних операцій можуть бути відсутні певні сигнали або, навпаки, присутні додаткові показники залежно від використовуваного обладнання та клінічної практики.

Оскільки до подальшого аналізу можуть бути залучені лише випадки, для яких повністю представлені всі обрані показники, розширення набору вхідних ознак безпосередньо призводить до скорочення обсягу придатних спостережень.

Таблиця 3.1 – Клініко-демографічна характеристика VitalDB

	Загальна хірургія (n = 4,930)	*Торакальна хірургія (n = 1,111)*	*Гінекологія (n = 230)*	*Урологія (n = 117)*	*Всього (n = 6,388)*
Демографічні дані					
Стать (ч)	2,524 (51.2%)	618 (55.6%)	0 (0%)	101 (86.3%)	3,243 (50.8%)
Вік (роки)	59 (48-68)	61 (52-70)	45 (35-55)	64 (58-72)	59 (48-68)
Зріст (см)	162 (156-169)	163 (156-169)	159 (155-163)	168 (161-173)	162 (156-169)
Вага (кг)	60 (53-69)	61 (54-69)	59 (53-66)	69 (62-77)	61 (53-69)
Хірургічний підхід					
Відкрита	3,104 (63.0%)	190 (17.1%)	65 (28.3%)	6 (5.1%)	3,368 (52.7%)
Відеоскопічна	1,691 (34.2%)	889 (80.0%)	140 (60.9%)	34 (29.1%)	2,701 (42.3%)
Роботичована	135 (2.7%)	32 (2.9%)	25 (10.9%)	77 (65.8%)	269 (4.2%)
Анестезія					
Загальна	4,630 (93.9%)	1,093 (98.4%)	203 (88.3%)	117 (100.0%)	6,043 (94.6%)
Спінальна	246 (5.0%)	0 (0.0%)	27 (11.7%)	0 (0.0%)	273 (4.3%)
Седація	54 (1.1%)	18 (1.6%)	0 (0.0%)	0 (0.0%)	72 (1.1%)
Тривалість анестезії (хв)					
	150 (90-245)	170 (110-220)	135 (95-185)	190 (79-231)	150 (90-240)
Анастетик					
Севофлуран	2,076 (42.1%)	255 (23.0%)	172 (74.8%)	34 (29.1%)	2,537 (39.7%)
Десфлуран	1,012 (20.5%)	226 (20.3%)	69 (30.0%)	106 (90.6%)	1,383 (21.6%)
Пропофол	2,490 (50.5%)	996 (89.6%)	5 (2.2%)	7 (6.0%)	3,498 (54.8%)
Реміфентаніл	3,639 (73.8%)	1,000 (90.0%)	110 (47.8%)	86 (73.5%)	4,835 (75.7%)

Для формування узгодженого набору даних було використано бібліотеку Python vitaldb, яка дозволяє здійснювати вибірку підмножини випадків із наявністю заданих параметрів. У межах даної роботи було обрано ключові інтраоперативні показники, представлені у вигляді часових рядів. До них належать показники введення лікарських засобів, зокрема швидкість інфузії

пропофолу та реміфентанілу, а також основні фізіологічні параметри пацієнта: середній артеріальний тиск, насичення крові киснем (SpO_2), індекс біспектрального аналізу (BIS), що характеризує глибину анестезії, та концентрація вуглекислого газу в кінці видиху ($EtCO_2$), яка відображає ефективність вентиляції.

Після фільтрації бази даних за наявністю всіх зазначених показників було отримано 1823 клінічні випадки. На наступному етапі ці дані було поєднано з інформацією про пацієнтів та операційний контекст. До характеристик хірургічного втручання належать тип та профіль операції, хірургічний підхід, а також ознака екстреності. Персональні характеристики пацієнта включають стать, вік, індекс маси тіла, масу тіла, клас фізичного статусу за шкалою ASA, оцінку складності інтубації за шкалою Кормака, а також наявність супутніх захворювань артеріальної гіпертензії та цукрового діабету.

Окрім клінічних характеристик, до набору статичних властивостей було включено передопераційні лабораторні показники, що відображають функціональний стан основних фізіологічних систем організму. Оцінки функції нирок: рівень креатиніну, азоту сечовини крові та швидкість клубочкової фільтрації. Загальний стан пацієнта додатково описувався показниками гемоглобіну, гематокриту, альбуміну та глюкози крові, показникам згортання крові. Випадки, для яких була відсутня значна частина цих даних, були виключені з подальшого аналізу. У результаті такого відбору остаточний набір даних склав 1454 пацієнти.

Сформований таким чином датасет містить узгоджену інформацію про стан пацієнта під час операції разом із відповідними діями лікаря у вигляді введення анестетиків, що робить його придатним для подальшого використання у задачах машинного навчання, зокрема для моделювання процесу прийняття рішень в анестезіології.

3.2 Підготовка даних

Після завантаження набору клінічних випадків із бази VitalDB було виконано обробку даних, спрямовану на підготовку їх до подальшого використання в моделях машинного навчання.

Спочатку було виконано очищення даних від аномальних значень. Значення фізіологічних показників, що виходили за межі фізіологічно можливих діапазонів такі, як надто низькі або високі показники артеріального тиску чи частоти серцевих скорочень, були позначені як пропущені. Такий підхід дозволяє уникнути впливу артефактів вимірювання або технічних збоїв на результати моделювання.

Наступним кроком стала обробка пропущених значень. Для безперервних фізіологічних сигналів було застосовано лінійну інтерполяцію. Для параметрів, пов’язаних із введенням лікарських засобів, було використано метод перенесення останнього відомого значення вперед у часі, що відповідає реальній логіці роботи інфузійних помп, де швидкість введення залишається сталою до моменту її зміни.

Далі було виконано трансформацію лабораторних показників у формат, придатний для об’єднання з іншими джерелами інформації. Початкові лабораторні дані, представлені у вигляді окремих записів для кожного аналізу, були перетворені у широку таблицю, де кожному клінічному випадку відповідає один рядок.

Паралельно було підготовлено клінічні дані пацієнтів, які містили демографічні характеристики, інформацію про супутні захворювання та параметри оперативного втручання. Категоріальні змінні були перетворені у набір бінарних ознак за допомогою унітарного кодування. Бінарні параметри, такі як стать, наявність гіпертонії, діабету або екстрений характер операції, були представлені у числовій формі.

Було виконано порівняльний аналіз розподілів часових послідовностей інтраопераційних сигналів залежно від тривалості оперативного втручання. Для кожного клінічного випадку було визначено кількість записів у часовому ряді, що дозволило ідентифікувати операції з тривалістю понад 8 годин. Дані було розділено за цим принципом. Для кожної групи було розраховано описові статистики швидкостей введення препаратів, включно із середнім значенням, модою та показниками варіативності. Отримані результати продемонстрували відсутність суттєвих відмінностей між групами, що свідчить про те, що рівень ін'єкційних швидкостей не має значної залежності від загальної тривалості операції.

Після цього було виконано пошук некоректних значень у швидкості введення пропофолу, зокрема надто високих показників, що не можуть відповідати реальним клінічним умовам. У результаті було виявлено випадок, у якому зафіксоване значення швидкості введення становило 1162693, що однозначно є помилковим записом. Для порівняння, друге за величиною значення у вибірці становило лише 1200, що підтверджує аномальний характер виявленого показника та його невідповідність реальним фізіологічним межам.

Далі було сформовано набір часових ознак на основі фізіологічних сигналів і параметрів введення препаратів. Для кожного з основних сигналів було обчислено похідні ознаки, що відображають зміну значення між сусідніми часовими кроками. Такий підхід дозволяє моделі враховувати не лише поточний стан пацієнта, але й динаміку його змін.

Окремо були сформовані накопичувальні ознаки для препаратів. Для пропофолу та реміфентанілу було розраховано кумулятивну дозу як суму швидкостей введення за часом. Оскільки абсолютні значення накопиченої дози можуть суттєво відрізнятися між випадками, ці параметри були додатково нормалізовані логарифмічним перетворенням. Аналогічно було створено ознаку нормалізованого часу операції, яка відображає тривалість перебігу наркозу в логарифмічному масштабі.

Для більш точного відображення режиму введення препаратів було введено ознаки, що описують час від останнього болюсного введення. Було визначено порогове значення швидкості інфузії, яке інтерпретується як болюс. Для кожного моменту часу обчислювався інтервал від останнього такого епізоду. Отримані значення також були логарифмічно нормалізовані, результат нормалізації для одного з клінічних випадків наведено на рисунку 3.1. Ці характеристики дозволяють моделі відстежувати фазу після введення болюсу та потенційний фармакодинамічний ефект препарату.

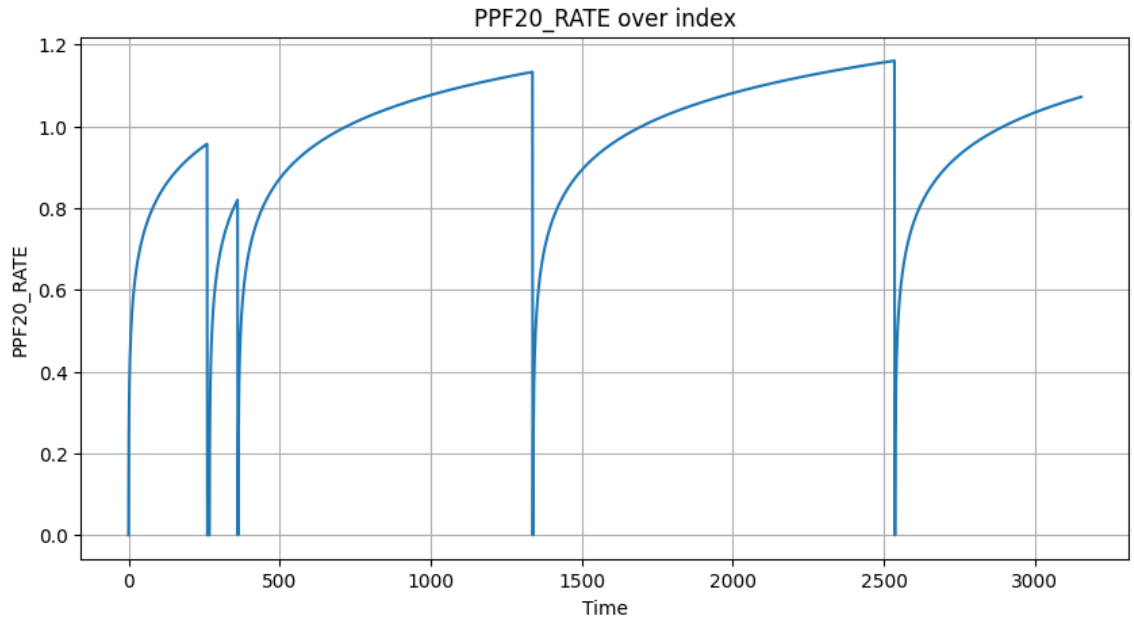


Рисунок 3.1 – Логарифмічно нормалізований час від останнього болюсного введення пропофолу для одного клінічного випадку

Після формування повного набору ознак було виконано масштабування даних. Для фізіологічних сигналів та їх похідних використовувалась стандартизація. Для швидкостей введення препаратів та їх змін застосовувалася робастна нормалізація. Лабораторні та числові клінічні показники також були стандартизовані.

Для подальшого навчання моделей було сформовано структуру вибірок із розділенням на тренувальну, валідаційну та тестову підмножини на рівні клінічних випадків.

Наступним етапом стало створення навчальних прикладів на основі ковзного вікна. Для кожного випадку формувались часові фрагменти фіксованої довжини, що відповідають останнім п’яти хвилинам спостережень, після чого прогнозувалась цільова змінна на наступний часовий горизонт тривалістю одну хвилину. Цільові значення визначались як максимальна швидкість введення пропофолу та реміфентанілу в межах прогнозного інтервалу, що дозволяє моделі передбачати пікову потребу у препаратах.

На фінальному етапі обробки всі сформовані набори даних були збережені у вигляді окремих структур: статичні характеристики пацієнтів, часові ряди сигналів, а також індекси тренувальної, валідаційної та тестової вибірок.

3.3 Побудова структури моделі

Задача визначення дози анестетиків є нетривіальною з точки зору машинного навчання: модель має одночасно опрацьовувати різнорідні джерела даних статичну інформацію про пацієнта та динамічні інтраопераційні часові ряди і на їхній основі формувати безперервний регресійний вихід. Кожна з розглянутих нижче архітектур підходить до цієї задачі з різної точки зору і стикається з властивими їй труднощами.

Перед вибором конкретної архітектури моделі необхідно визначити спосіб інтеграції статичних та часових компонентів вхідних даних, оскільки саме це рішення визначає структуру всієї подальшої побудови моделі. У

задачах прогнозування фізіологічних параметрів пацієнта зазвичай розглядають два принципові підходи: ранню та пізню інтеграцію.

Рання інтеграція передбачає об’єднання статичних характеристик пацієнта з часовими даними шляхом конкатенації. Це означає, що один і той самий статичний вектор додається до вхідного вектора на кожному кроці часової послідовності. Перевагою такого підходу є простота реалізації, оскільки модель отримує єдиний потік даних і не потребує додаткових блоків для обробки різних типів ознак. Недоліком такого підхходу є штучне збільшення вхідного вектора. Це збільшує розмірність вхідних даних і ускладнює для моделі відокремлення впливу базових характеристик пацієнта від впливу динамічних інтраопераційних сигналів. Також оскільки є значно більше статичних змінних, ніж часових, модель буде значну частину свого часу взаємодіяти зі статичним, незмінюючимися в часі даними.

Пізня інтеграція базується на роздільному кодуванні різних типів інформації. Статичні дані обробляються окремим модулем, який формує узагальнене представлення пацієнта і операції. Часові ряди інтраопераційних сигналів обробляються окремою моделлю послідовностей, яка формує опис поточного стану. Після цього обидва представлення об’єднуються безпосередньо перед фінальним прогнозом. Такий підхід є складнішим в реалізації, однак дозволяє кожному модулю спеціалізуватись на своєму типі даних.

У контексті задачі прогнозування стану пацієнта під час анестезії пізня інтеграція є більш обґрунтованою. Статичні параметри визначають загальний фізіологічний профіль пацієнта та його базові ризики, тоді як інтраопераційні сигнали відображають швидкі зміни стану організму в режимі реального часу. Ці типи інформації мають різну природу, різні масштаби варіації та різні часові властивості. Роздільна обробка статичних і динамічних ознак дозволяє моделі явно враховувати цю відмінність, що є важливим для підвищення стабільності прогнозу та покращення інтерпретованості результатів.

Запропонована модель побудована за принципом пізньої інтеграції, де статичні та часові дані обробляються незалежними модулями, а їх поєднання відбувається лише на рівні латентних представлень. Такий підхід дозволяє окремо враховувати базові характеристики пацієнта та його поточний фізіологічний стан під час операції.

Таким чином, структура моделі включає такі основні блоки як представлено нижче.

1. Кодувальник часових даних – модуль, який обробляє послідовність інтраопераційних сигналів (фізіологічні параметри та поточні значення моніторингу) та формує узагальнене латентне представлення динамічного стану пацієнта.
2. Кодувальник статичних даних – модуль, який перетворює демографічні та лабораторні показники (вхідні характеристики, що не змінюються протягом операції) у компактний вектор представлення пацієнта.
3. Блок злиття – механізм об’єднання латентних представлень часових та статичних даних. На цьому етапі формується єдиний вектор, який одночасно відображає поточний стан пацієнта та його індивідуальний фізіологічний профіль.
4. Спільний блок обробки – проміжний модуль, який додатково обробляє інтегроване представлення та формує загальний контекст для задачі прогнозування дозування препаратів.
5. Два незалежні вихідні регресійні блоки – фінальні підмережі, кожна з яких відповідає за прогноз окремого параметра. Модель формує два виходи прогноз дозування пропофолу, прогноз дозування реміфентанілу.

Загальна логіка роботи моделі полягає в тому, що часові сигнали визначають поточну динаміку стану організму, тоді як статичні характеристики моделюють індивідуальні особливості пацієнта. Їх поєднання в латентному

просторі дозволяє підвищити точність прогнозу. Загальна архітектура системи наведена на рисунку 3.2.

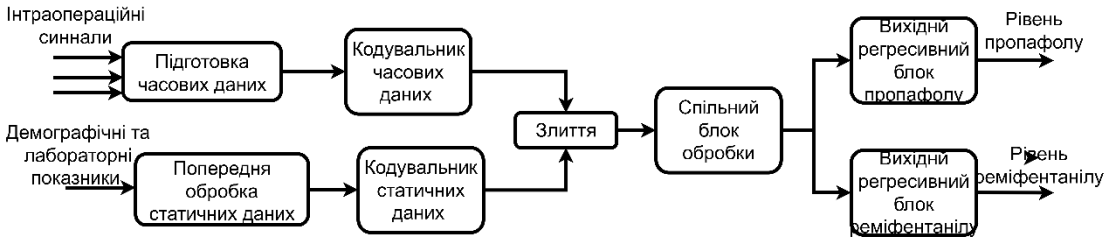


Рисунок 3.2 – Архітектура моделі прогнозування дозування анестетиків

3.4 Топологія нейронних мереж

3.4.1 Кодувальник статичних даних

Архітектура моделі складається з послідовно з’єднаних функціональних блоків, кожен з яких виконує окрему роль у формуванні кінцевого прогнозу дозування препаратів.

В якості кодувальника статичних даних (рис. 3.3) у запропонованій архітектурі використовується модуль, побудований на основі багатошарового перцептрона з залишковими зв’язками та механізмом сигмоїдального керування ознаками. На першому етапі вхідні ознаки проєктуються у прихований простір розмірності 256 за допомогою лінійного шару та Layer Normalization. Після цього представлення проходить через залишковий блок, що складається з двох повнозв’язних шарів із функцією активації RELU та Dropout-регуляризацією.

Використання залишкових зв’язків дозволяє мережі не втрачати початкову інформацію під час проходження через глибші шари та полегшує оптимізацію моделі. Додатково застосовується механізм сигмоїдального

керування, який формує вагові коефіцієнти для кожної компоненти прихованого вектора. Таким чином модель може автоматично підсилювати найбільш інформативні статичні ознаки та пригнічувати менш важливі параметри.

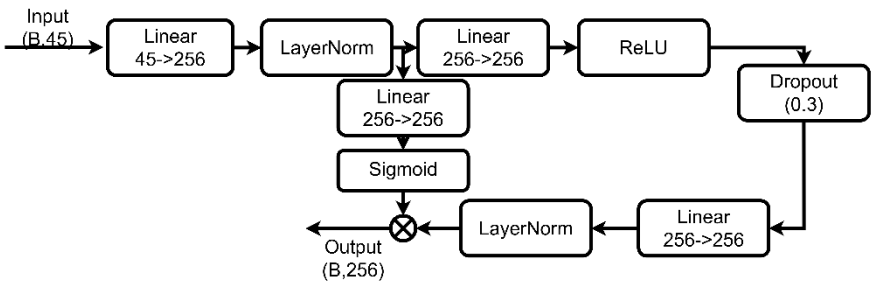


Рисунок 3.3 – Архітектура кодувальника статичних даних

3.4.2 Кодувальник часових даних

Кодувальник часових даних відповідає аналізу наслідків фізіологічних сигналів та історії введення препаратів у часовому вікні спостереження. Основною задачею цього модуля є перетворення багатовимірної часової хвороби в прихований вектор, який узагальнює динаміку стану пацієнта перед моментом прогнозування.

У моделі передбачено підтримку трьох альтернативних архітектури кодувальника часових даних: рекурентної мережі LSTM із механізмом уваги, трансформеру та Informer кодувальнику.

LSTM із часовим механізмом уваги наведено на рисунку 3.4. У даній архітектурі використовуються приховані стани, що формуються для кожного моменту часу під час обробки послідовності. Їх узагальнення виконується за допомогою агрегуючого механізму уваги. Отримане представлення надалі

передається до лінійного шару та шару нормалізації, де виконується фінальне перетворення ознак.

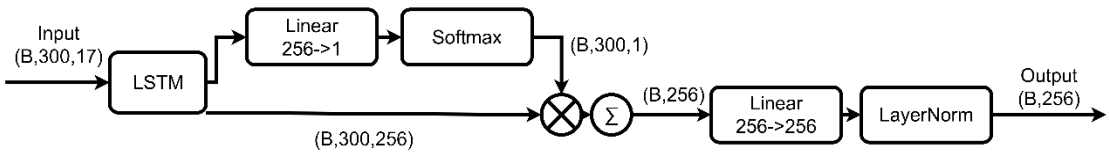


Рисунок 3.4 – Архітектура LSTM кодувальника часових даних

Архітектура трансформера для обробки часових послідовностей показана на рисунку 3.5. На вході модель отримує послідовність векторів ознак, які спочатку проєктуються у внутрішній простір фіксованої розмірності. Після цього до кожного елемента послідовності додається позиційна інформація, що задає його місце у часі.

Далі дані послідовно проходять через стек шарів трансформерного кодувальника. На кожному шарі виконується обробка послідовності з урахуванням взаємозв’язків між усіма часовими кроками за допомогою механізму багатоголової самоуваги, після чого застосовується багатошарова повнозв’язна мережа, яка виконує нелінійне перетворення ознак. Структура кожного шару має механізмами нормалізації та використовує залишкові зв’язки.

Після проходження всіх шарів отримується набір представлень для кожного моменту часу. Далі виконується агрегування по часовій осі за допомогою зваженого сумування з навчуваними вагами, в результаті чого формується єдине векторне представлення всієї послідовності. Що надалі передається до повнозв’язного шару та шару нормалізації, де виконується фінальне перетворення ознак.

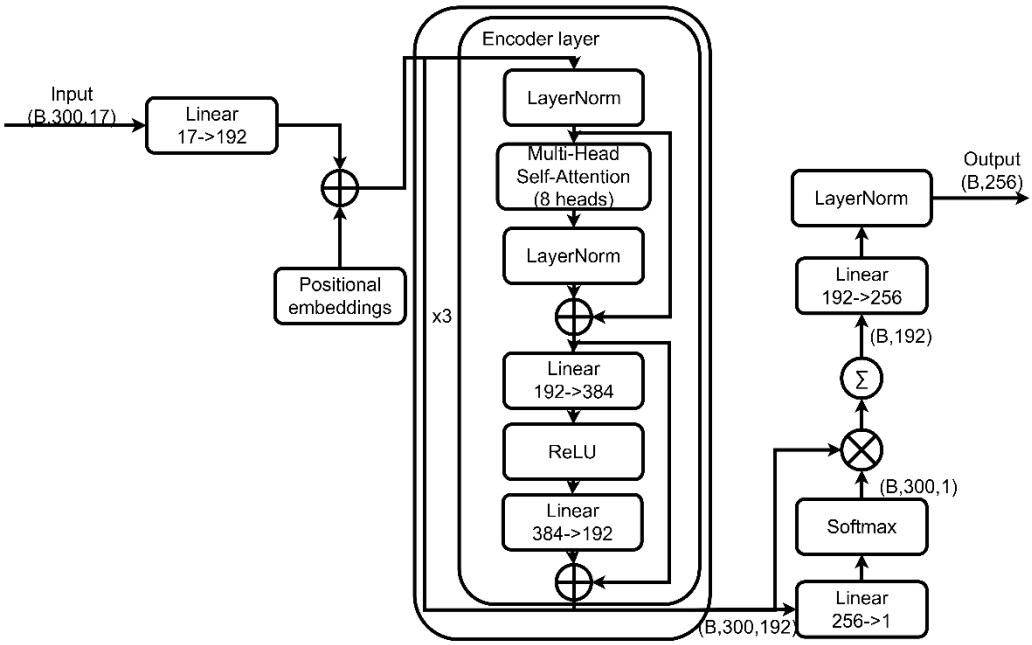


Рисунок 3.5 – Архітектура трансформера кодувальника часових даних

На відміну від класичної архітектури трансформера, описаної вище, модель Informer орієнтована на ефективну обробку довгих часових послідовностей та зменшення обчислювальної складності механізму уваги. Архітектура Informer для аналізу часових рядів показана на рисунку 3.6.

Як і у випадку трансформера, на вході модель отримує послідовність векторів ознак, які спочатку перетворюються лінійним шаром у прихований простір фіксованої розмірності. Після цього до отриманих представлень додається позиційна інформація, яка дозволяє моделі враховувати порядок елементів у часовій осі.

Далі послідовність проходить через стек «енкодерних шарів», однак замість стандартної багатоголової самоуваги використовується розріджена багатоголова самоувага, яка обчислює увагу лише для найбільш інформативних часових позицій. Такий підхід дозволяє зберігати здатність моделі знаходити глобальні залежності між віддаленими моментами часу, одночасно зменшуючи обсяг необхідних обчислень.

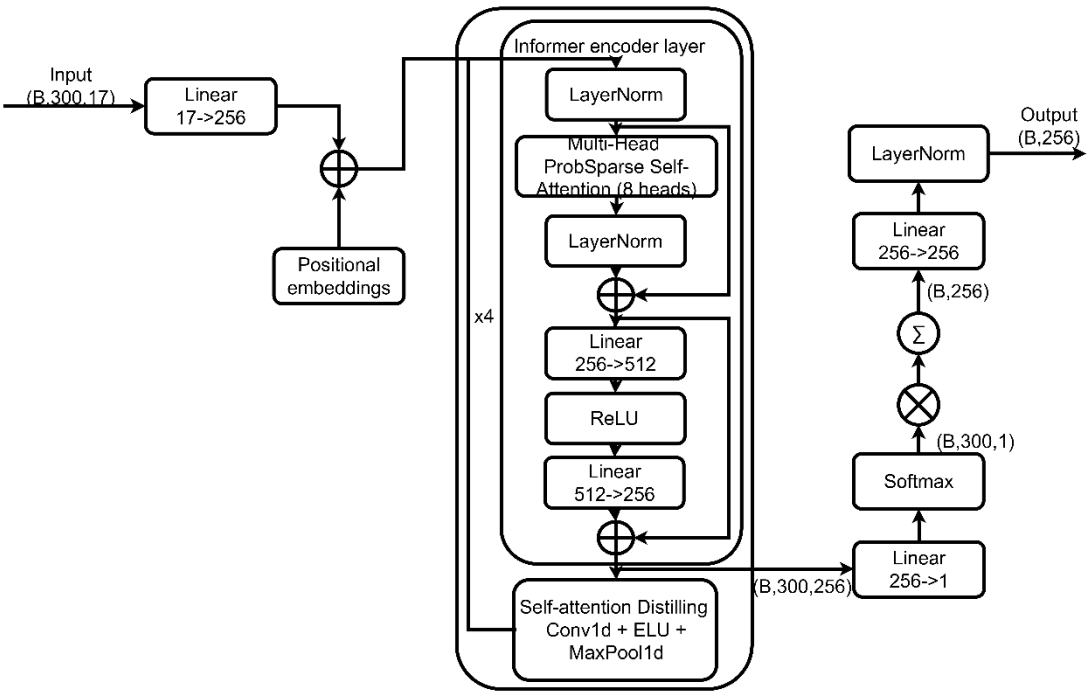


Рисунок 3.6 – Архітектура Informer кодувальника часових даних

Крім того, між окремими шарами кодувальника використовується механізм дистиляції, який поступово скорочує довжину послідовності, агрегуючи інформацію та формуючи більш компактні представлення. Це дає змогу моделі ефективніше концентруватися на ключових закономірностях часових даних та формувати більш узагальнений прихований контекст.

Після проходження всіх шарів Informer кодувальнику формується набір контекстно збагачених прихованих представлень, у яких зберігається глобальна інформація про динаміку послідовності. На завершальному етапі виконується агрегування по часовій осі, в результаті чого отримується єдиний вектор.

У порівнянні з класичним трансформером, Informer має додаткові механізми ефективної уваги та скорочення послідовності, що дозволяє йому працювати з більшими часовими вікнами та формувати більш глибокий прихований контекст усередині моделі.

3.4.3 Блоки обробки об’єднаного представлення і регресії рівня пропофолу і реміфентанілу

Отримані вектори ознак динамічних та статичних властивостей об’єднуються за допомогою конкатенації. Після цього вони обробляються спільним блоком обробки (рис. 3.7).

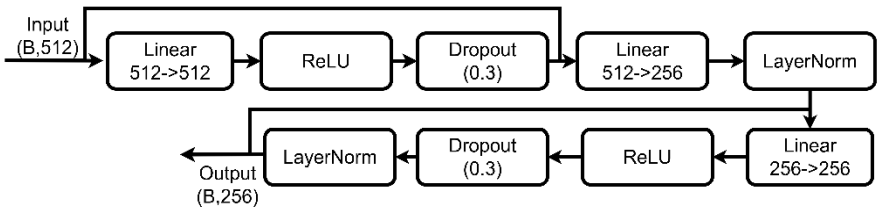


Рисунок 3.7 – Архітектура спільного блоку обробки

Спільний блок обробки складається з трьох послідовних підблоків. На вхід подається об’єднане представлення вхідних даних.

Перший підблок є повнозв’язним шаром, що зберігає розмірність простору ознак, з функцією активації ReLU та Dropout із ймовірністю 0.3. Його вихід передається далі з використанням пропускного з’єднання.

Другий підблок також реалізовано у вигляді повнозв’язного шару, який зменшує розмірність ознак у два рази.

Третій підблок складається з повнозв’язного шару, що зберігає розмірність простору ознак, з функцією активації ReLU та Dropout із ймовірністю 0.3, після чого застосовується шар нормалізації. Його вихід додається до результату другого підблоку через залишкове з’єднання, формуючи фінальне представлення блоку.

Архітектура вихідного регресійного блоку показана на рисунку 3.8. Вона однакова як для пропофолу так і для реміфентанілу.

Вектор ознак послідовно проходить через триповнозв’язні шари, кожен з який зменшує розмір в 2 рази. Використовується функція активації ReLU і перед останнім шансом є 30% регуляризація Dropout.

На виході модель формує одне скалярне значення, що відповідає прогнозованій швидкості інфузії препарату.

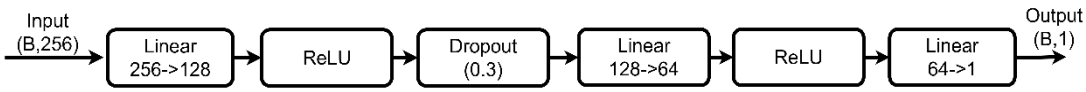


Рисунок 3.8 – Архітектура вихідного регресійного блоку

3.5 Методи навчання

Процес навчання моделей організовано з урахуванням специфіки клінічних даних, зокрема нерівномірного розподілу цільових значень, наявності рідкісних болюсних введень та необхідності уникнення витоку інформації через часову структуру даних. Також важливим аспектом була забезпеченість коректного порівняння результатів між різними архітектурами моделей.

В якості функції втрат використовувалась модифікована версія Focal Loss. Вона дозволяє збільшити внесок складних та рідкісних прикладів у процес оптимізації, зокрема болюсних введень препаратів, які представлені у даних значно рідше за стабільні рівні. Порогове значення для визначення «пікових станів» було встановлено на рівні 3.0. І для додаткового покращення здатності моделі виявляти різкі переходи між постійним рівнем введення та болюсами до функції втрат було введено окремий штрафний компонент. Його суть в тому, що якщо модель повернула менше за порогове значення, коли цільова змінна показує «піковий стан», то значення функції втрат для цього

передбачення множиться на ваговий коефіцієнт. Для пропофолу використовувався ваговий коефіцієнт 8, а для реміфентанілу – 10, що дозволило підсилити вплив помилок під час прогнозування пікових введень.

Для запобігання перенавчанню застосовувалась рання зупинка навчання. Навчання припинялось у випадку відсутності покращення значення функції втрат на валідаційній вибірці протягом п’яти послідовних перевірок.

В якості оптимізатора використовувався Adam із початковою швидкістю навчання 3e-4 і коефіцієнтом вагового розпаду 1e-5. В якості планувальника застосувалася косинусна стратегія зміни швидкості навчання з періодичним перезапуском, що забезпечувало стабільніший процес оптимізації. Розмір пакета під час навчання становив 512 прикладів.

Оцінювання якості моделей виконувалось за допомогою середньої абсолютної похибки, кореня середньоквадратичної похибки, коефіцієнта детермінації та середнього зміщення прогнозу. Окремо аналізувалась підмножина рідкісних пікових значень цільової змінної. Для цієї підмножини додатково обчислювались спеціалізовані метрики похибки, а також показники точності та повноти виявлення пікових введень.

3.6 Побудова інтелектуальної системи

Інтелектуальна система на основі моделі машинного навчання є ширшою структурою, ніж сама модель прогнозування. Модель виступає центральним обчислювальним компонентом, однак її робота потребує організації процесу отримання, підготовки та передачі даних у форматі, придатному для обробки нейронною мережею.

У межах такої системи зовнішні джерела даних передають інформацію про стан пацієнта, параметри моніторингу та історію введення препаратів. За

необхідності система може виконувати запит додаткових даних, які потрібні для формування коректного вхідного представлення для моделі. Таким чином інтелектуальна система забезпечує перехід від первинних клінічних даних до структурованого набору ознак, який використовується моделлю машинного навчання.

Оскільки модель працює з часовими рядами, система повинна підтримувати механізми накопичення та оновлення часового контексту. Для цього система повинна неперервно формувати структуру даних, яка може бути використана як вхідні дані моделі. Дані для моделі формуються у вигляді ковзних часових вікон, що означає інтелектуальна система повинна зберігати дані за останні 15 хвилин. Також, оскільки результати надаються не в п’яти секундних інтервалах система повинна активно інтерпретувати показники і готувати їх для моделі.

Результатом роботи моделі є прогноз швидкості інфузії препаратів. Отриманий прогноз може використовуватись як допоміжний елемент системи підтримки прийняття рішень та надаватись медичному персоналу для подальшого аналізу. При цьому остаточне рішення щодо введення препаратів залишається за спеціалістом.

У межах даного дослідження основна увага приділяється саме побудові та навчанню моделі машинного навчання як центрального компонента інтелектуальної системи.

Висновки до розділу 3

У третьому розділі описано повний побудову інтелектуальної системи визначення препаратів для анестезії.

Для дослідження було обрано відкриту базу даних VitalDB, що містить багатопараметричні інтраопераційні записи 6388 хірургічних пацієнтів. Після послідовної фільтрації за наявністю ключових фізіологічних сигналів і передопераційних лабораторних показників сформовано узгоджений датасет із 1454 клінічних випадків, достатній для побудови та оцінки моделей машинного навчання.

На етапі підготовки даних виконано очищення аномальних вимірювань, лінійну інтерполяцію пропущених фізіологічних сигналів та перенесення останнього відомого значення для параметрів інфузії. Сформовано розширений набір ознак, що включає похідні динамічні характеристики сигналів, кумулятивні дози препаратів із логарифмічною нормалізацією, ознаки часу від останнього болюсного введення, а також унітарне кодування категоріальних змінних.

Запропоновано архітектуру моделі за принципом пізньої інтеграції, яка дозволяє роздільно обробляти статичний профіль пацієнта та динамічні інтраопераційні сигнали. Статичний кодувальник побудовано на основі багатошарового перцептрона із залишковими. Для обробки часових рядів передбачено три альтернативні архітектури кодувальника: LSTM із механізмом уваги, трансформер та Informer. Об'єднані латентні представлення обробляються спільним блоком, після чого формуються два незалежні регресійні виходи – для рівня дозування пропофолу та реміфентанілу.

Описано концептуальну архітектуру інтелектуальної системи загалом, де модель виступає центральним обчислювальним компонентом системи підтримки прийняття рішень, що забезпечує формування прогнозів для медичного персоналу без підміни клінічного рішення лікаря.

РОЗДІЛ 4. ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ І ПРОВЕДЕННЯ ЕКСПЕРИМЕНТУ

4.1 Аналіз інструментарію розробки

Для розробки інтелектуальної системи було обрано мову програмування Python, що зумовлено її широким застосуванням у сфері аналізу даних та машинного навчання, а також наявністю розвиненої екосистеми бібліотек для обробки та моделювання медичних часових рядів.

Центральним аспектом проєктування програмної архітектури став вибір фреймворку глибокого навчання. За результатами аналізу сучасних програмних рішень було обрано бібліотеку PyTorch, яка реалізує підхід динамічного формування обчислювального графа під час виконання. Така особливість забезпечує підвищену гнучкість при розробці складних моделей та дозволяє адаптувати архітектуру до специфіки задачі без необхідності попереднього фіксування структури обчислень.

Динамічна природа PyTorch є особливо важливою в контексті даного дослідження, оскільки розроблювана система передбачає обробку як статичних характеристик пацієнтів, так і динамічних часових рядів клінічного моніторингу. Це створює передумови для реалізації мультимодальних архітектур із різнорідними входами та диференційованими функціями втрат, що враховують особливості окремих типів даних.

Крім того, суттєвою перевагою PyTorch є можливість глибокої кастомізації моделей. З огляду на специфіку медичних задач та необхідність адаптації стандартних архітектур до предметної області анестезіології, важливим є доступ до низькорівневих тензорних операцій. Це забезпечує можливість реалізації спеціалізованих компонентів моделі, зокрема модифікованих механізмів уваги та методів часового кодування.

Додатково до програмного стеку включено бібліотеку NumPy, що використовується для ефективної реалізації числових та матричних операцій, а також бібліотеку Pandas, яка забезпечує інструменти для структурованої обробки та підготовки даних пацієнтів. Використання зазначених інструментів дозволяє забезпечити узгоджений та відтворюваний процес підготовки даних перед подальшим моделюванням.

4.2 Проведення експерименту

Експериментальне дослідження проводилося у хмарному середовищі Google Colab з використанням підписки Colab Pro. Для виконання обчислень застосовувався графічний прискорювач типу L4 у конфігурації High-RAM, що забезпечувало достатній обсяг відеопам’яті та оперативної пам’яті для роботи з великими часовими рядами та глибокими нейронними мережами.

На початковому етапі виконувалося підключення до Google Drive, де зберігалися вихідні дані та попередньо сформовані кешовані набори даних. Після завантаження сирих даних у пам’яті формувався повний набір даних на основі механізму рухомого вікна розміром 300 часових кроків. Такий підхід дозволяв перетворити безперервні часові ряди фізіологічних параметрів у набір незалежних навчальних прикладів, придатних для подачі до моделі машинного навчання.

Для оптимізації часу підготовки даних використовувався механізм кешування. Якщо файл вже існував у Google Drive, система завантажувала його без повторної генерації. У випадку відсутності набір даних формувався з нуля на основі початкових записів моніторингу. Такий підхід суттєво скорочував час повторних експериментів та дозволяв уникнути повторної обробки багатогігабайтних часових рядів.

У результаті формування набору даних було отримано:

- 1 426 122 навчальних приклади;
- 256 786 прикладів для валідації;
- 256 786 прикладів для тестування.

Загальний обсяг сформованих даних становив близько 5 ГБ. Через обмеження обсягу відеопам’яті графічного процесора лише валідаційний набір повністю завантажувався у GPU-пам’ять. Навчальний та тестовий набори зберігалися в оперативній пам’яті та передавалися до моделі поступово під час виконання навчання.

Для зменшення споживання пам’яті навчальний набір додатково поділявся на 10 окремих груп. Навчання моделі проводилося послідовно по цих частинах, що дозволяло уникнути перевищення доступного обсягу пам’яті GPU. Оцінювання моделі на валідаційному наборі виконувалося після обробки кожних п’яти груп, це забезпечувало контроль якості моделі і дозволяло зберігати інформацію про модель посередині епохи.

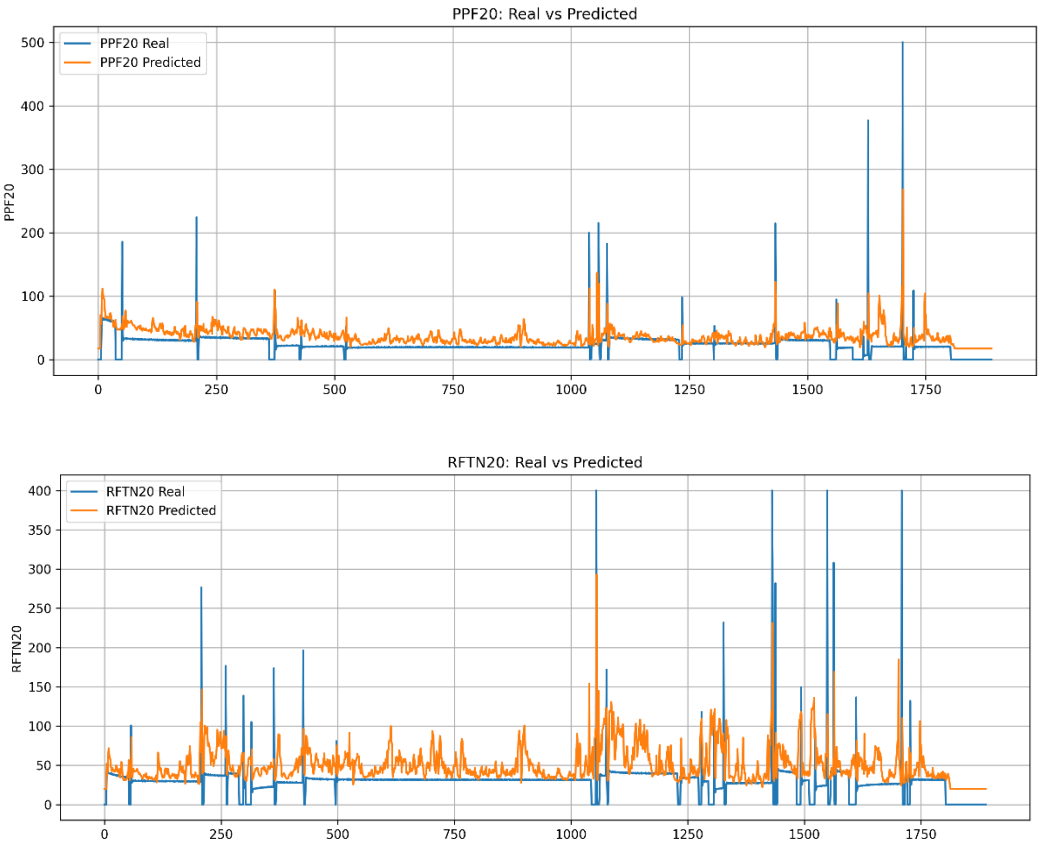
У процесі експериментів використовувалися декілька варіантів підготовлених наборів даних, що відрізнялися горизонтом прогнозування та способом формування цільової змінної. Для збереження даних застосовувався формат NPZ.

4.3 Результати експерименту

Для оцінки ефективності моделі запропонованої системи було проведено експеримент із використанням трьох архітектур нейронних мереж: LSTM, трансформер та Informer.

На рисунку 4.1 наведено результати роботи регресійної моделі з кодувальника часових даних архітектури LSTM. За результатами порівняння

прогнозів із реальними значеннями видно, що для пропофолу модель добре відтворює загальний тренд і базовий рівень дозування, проте часто згладжує або недооцінює короткочасні пікові підвищення. Для реміфентанілу LSTM частково повторює динаміку, однак у багатьох ділянках спостерігається завищення прогнозованого рівня та недостатня точність у відтворенні різких коротких змін, і видна чітка нестабільність моделі.

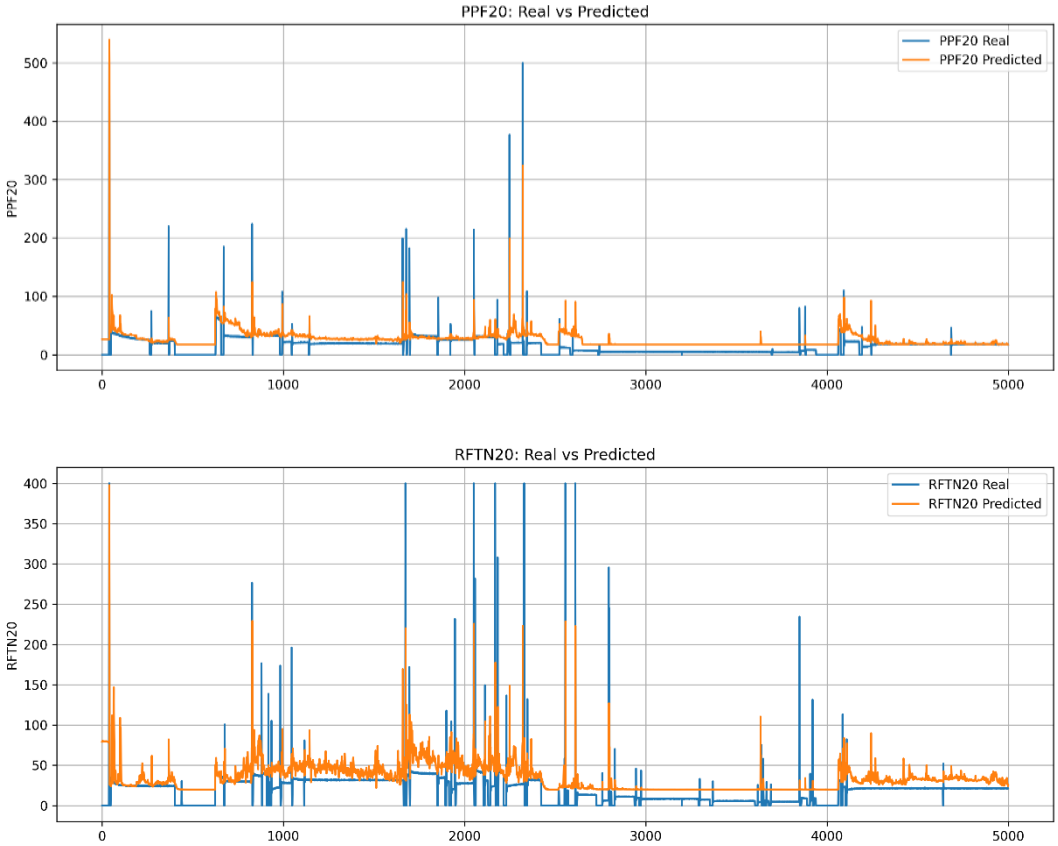


Помаранчева лінія показує рівень, запропонований системою,
синя – рівень, встановлений анестезіологом.
На верхньому графіку відображено зміну рівня ін'єкції пропофолу в
часі (секунди), на нижньому – реміфентанілу.

Рисунок 4.1 – Результати роботи регресійної моделі LSTM

На рисунку 4.2 представлено результати регресійної моделі Transformer. У процесі експериментів модель показала вищу складність навчання та менш стабільну поведінку під час прогнозування. Незважаючи на це, архітектура

трансформер продемонструвала здатність виявляти складні часові залежності у даних її результати виявилися найгіршими серед запропонованих часових кодувальників.

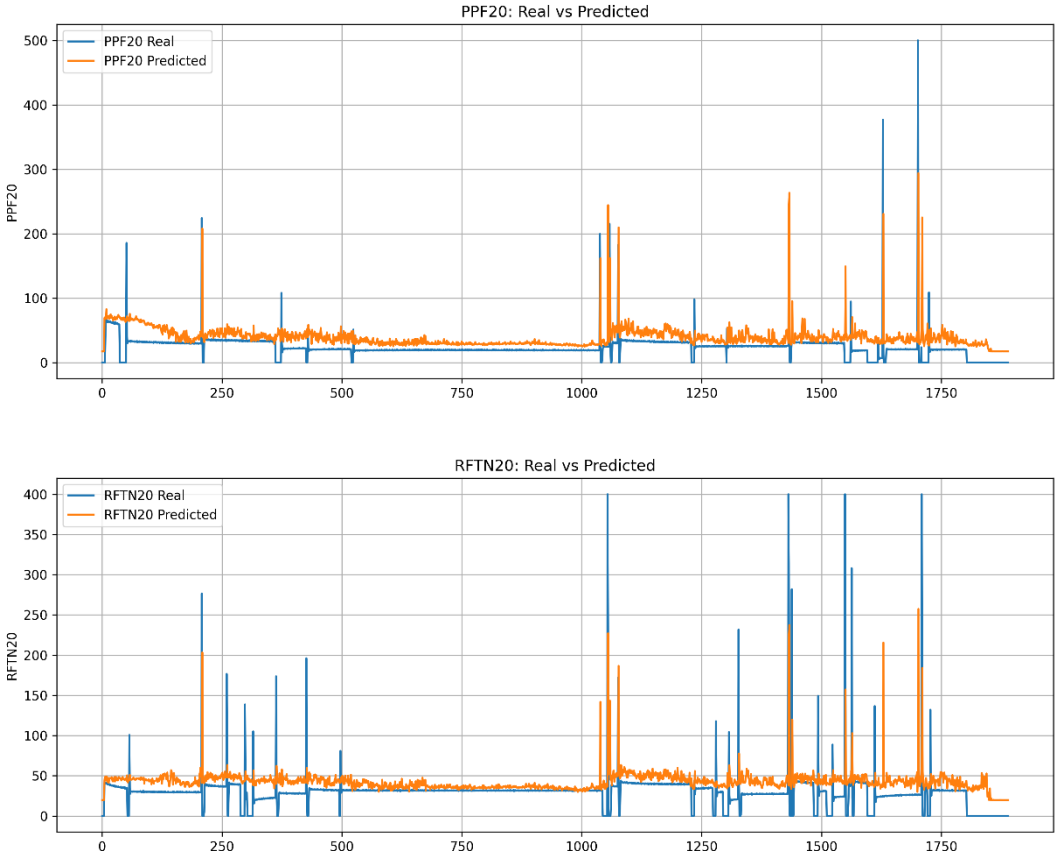


Помаранчева лінія показує рівень, запропонований системою,
синя – рівень, встановлений анестезіологом.
На верхньому графіку відображено зміну рівня ін'єкції пропофолу в
часі (секунди), на нижньому – реміфентанілу.

Рисунок 4.2 – Результати роботи регресійної моделі Transformer/

На рисунку 4.3 наведено результати роботи регресійної моделі Informer. Дана архітектура забезпечила найшвидший і найстабільніший процес навчання серед досліджених моделей. Модель показала найкращу якість прогнозування, передбачивши % ін'єкційного підвищення рівня препаратів. В той же момент

Informer показала найкращий у стабільному утриманні середнього рівня введеня препаратів.



Помаранчева лінія показує рівень, запропонований системою,
синя – рівень, встановлений анестезіологом.
На верхньому графіку відображено зміну рівня ін'єкції пропофолу в
часі (секунди), на нижньому – реміфентанілу.

Рисунок 4.3 – Результати роботи регресійної моделі Informer

У таблиці 4.1 наведено загальне порівняння трьох моделей за точністю прогнозування концентрацій пропофолу і реміфентанілу, здатністю до знаходження болюсних введень та швидкістю навчання.

Загалом, LSTM демонструє стабільну якість прогнозування, особливо для пропофолу, але потребує найбільших витрат часу на навчання. трансформер показує конкурентні результати за середньою похибкою та добре працює для

оцінки концентрацій, однак поступається за здатністю знаходити болюси реміфентанілу. Informer є найбільш збалансованою моделлю: він забезпечує високу точність знаходження болюсів для обох препаратів і при цьому має найкращу швидкість навчання.

Таблиця 4.1 – Результати експерименту

	LSTM	*Трансформер*	*Informer*
RMSE для пропофолу	1.7175	1.8072	1.7485
RMSE для реміфентанілу	1.5964	1.5234	1.4667
MAE для пропофолу	0.7752	0.7358	0.8237
MAE для реміфентанілу	0.9123	0.7477	0.8937
AUC-ROC для пропофолу	0.8148	0.8208	0.8963
AUC-ROC для реміфентанілу	0.7883	0.6712	0.8257
Швидкість навчання	26.4 хв/ епоха	23.4 хв/епоха	13.5 хв/епоха

Таким чином, Informer є найефективнішим варіантом з точки зору практичного використання, оскільки поєднує високу якість виявлення болюсів та значно швидше навчання порівняно з іншими моделями.

4.4 Взаємодія користувача з інтелектуальною системою

Взаємодія користувача з інтелектуальною системою реалізована через програмний API, побудований за архітектурним стилем REST. Такий підхід забезпечує непов’язаність клієнтської частини та серверу. Система реалізована на основі мікросервісної архітектури. Усі запити передаються у форматі JSON. Інтерфейс організований як послідовність логічно пов'язаних ендпоінтів, кожен з яких відповідає окремому етапу клінічного процесу.

На першому етапі у системі створюється запис пацієнта через відповідний ендпоінт реєстрації. До статичних характеристик, що передаються на цьому кроці, належать демографічні дані – вік, стать, антропометричні показники, – а також анамнестична інформація: наявність хронічних захворювань, відомі алергічні реакції та попередній досвід анестезії. Ці параметри формують базовий профіль пацієнта, який слугує незмінним контекстом для всіх подальших прогнозів у межах госпіталізації. Система повертає унікальний ідентифікатор пацієнта, що використовується як ключ прив'язки для всіх наступних записів.

Після реєстрації пацієнта до системи передаються результати лабораторних досліджень. Кожен аналіз оформлюється як окремий запис, прив'язаний до ідентифікатора пацієнта, і містить числові значення показників із зазначенням дати та часу забору матеріалу. Часова мітка є критично важливою, оскільки система враховує актуальність лабораторних даних при формуванні прогнозу.

Перед початком оперативного втручання створюється запис операції, що містить характеристики конкретного хірургічного випадку: тип та очікувана тривалість втручання, застосовувана техніка анестезії, позиціювання пацієнта та інші параметри, що визначають операційний контекст. Цей запис активує операційну сесію, у межах якої система переходить у режим обробки даних реального часу. З цього моменту всі наступні запити інтерпретуються системою як пов'язані з активним хірургічним випадком.

Під час проведення операції система у реальному часі отримує часові фізіологічні параметри з медичного обладнання. До таких параметрів належать показники гемодинаміки, рівень насичення крові киснем, глибина анестезії за індексом BIS, концентрація анестетика у видихуваному повітрі та інші змінні, що характеризують поточний стан пацієнта. Кожен пакет даних передається через відповідний ендпоінт із зазначенням часової мітки, що дозволяє системі реконструювати динаміку стану пацієнта і враховувати тренди змін, а не лише

миттєві значення. Частота передачі даних визначається клінічними вимогами і може варіюватися залежно від стабільності стану пацієнта.

При отримані запиту на прогноз дозування. Система формує вектори статичного профілю пацієнта, лабораторних показників, параметрів операції та поточних фізіологічних вимірювань, і передає їх до моделі, формуючи прогрноз, результат повертає через API у вигляді структурованої відповіді.

Загальна структура API системи представлена у вигляді інтерактивної документації Swagger (OpenAPI), яка відображає всі доступні ендпоінти, їх параметри та формати запитів і відповідей. На рисунку 4.9 наведено повний перелік REST-інтерфейсів системи.

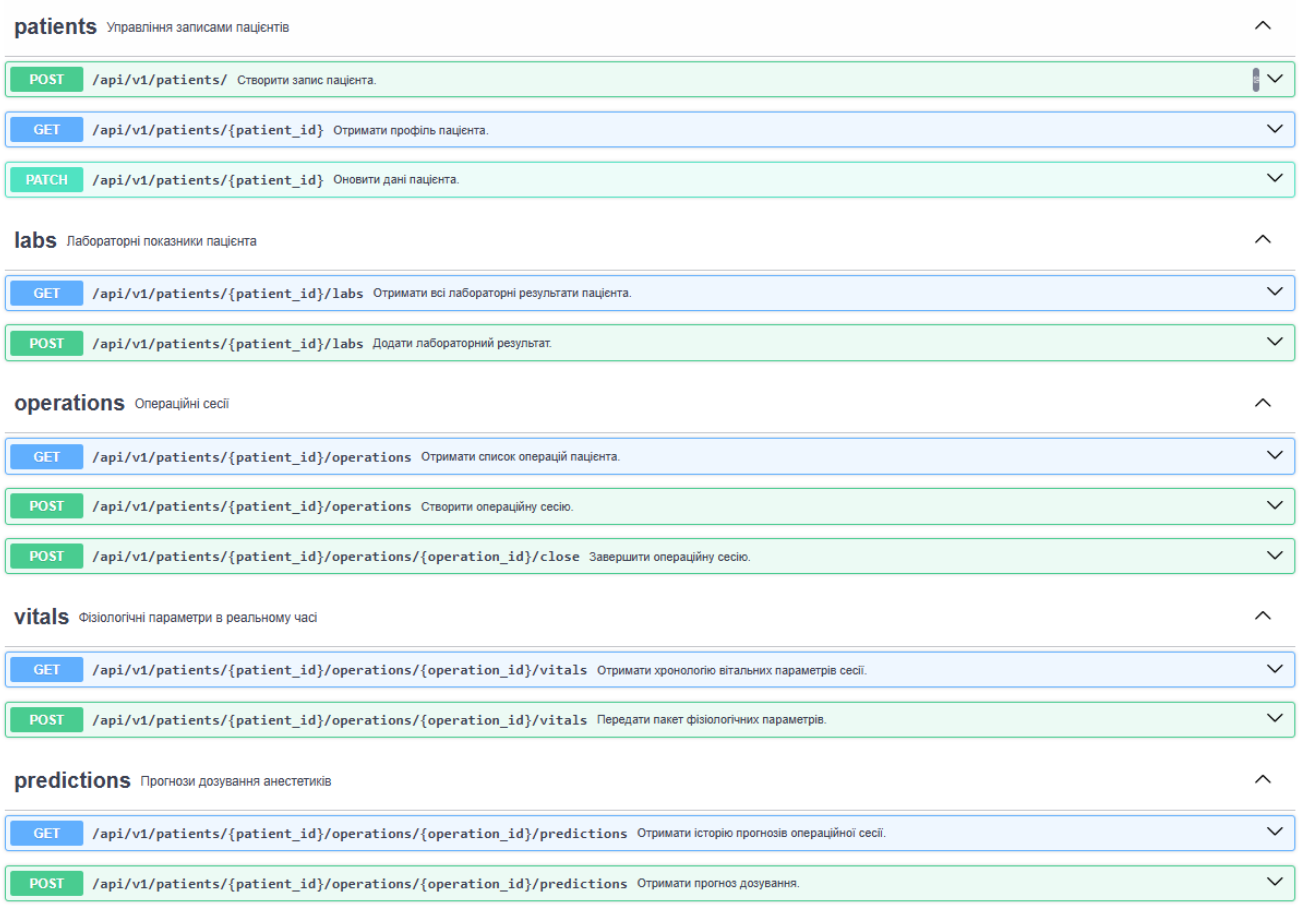


Рисунок 4.4 – Перелік REST-інтерфейсів інтелектуальної системи

Висновки до розділу 4

У четвертому розділі описано інструментарій розробки, проведення експериментального дослідження та аналіз отриманих результатів.

Для реалізації системи обрано мову програмування Python у поєднанні з фреймворком PyTorch, що забезпечило необхідну гнучкість при побудові мультимодальної архітектури та низькорівневий контроль над тензорними операціями. Експериментальне дослідження проводилося у хмарному середовищі Google Colab Pro з використанням GPU-прискорювача. Для ефективного управління обмеженими ресурсами відеопам'яті застосовувалися механізм кешування набору даних та поділ навчальної вибірки на рівні частини, що суттєво скорочувало час повторних експериментів.

За результатами порівняльного аналізу трьох архітектур нейронних мереж встановлено, що LSTM забезпечує стабільне відтворення загального тренду дозування, однак недостатньо точно реагує на короткочасні пікові зміни та потребує найбільших часових витрат на навчання. Базовий трансформер продемонстрував конкурентні результати за окремими метриками, але загалом показав найгіршу якість виявлення болюсних введень. Informer виявився найбільш збалансованою архітектурою: він забезпечив найвищу точність виявлення болюсів для обох препаратів, найнижчу похибку прогнозування реміфентанілу та найвищу швидкість навчання, що робить його оптимальним вибором для практичного застосування.

Взаємодія з системою реалізована через REST API із підтримкою формату JSON, що забезпечує технологічну сумісність із наявним медичним обладнанням та госпітальними інформаційними системами. Інтерфейс охоплює повний клінічний цикл від реєстрації пацієнта до отримання прогностичної рекомендації в реальному часі.

ВИСНОВКИ

У магістерській дисертації вирішено актуальну науково-практичну задачу автоматизованого підбору препаратів для анестезії на основі методів машинного навчання.

У першому розділі проведено аналіз предметної області анестезіології та обґрунтовано актуальність задачі автоматизованого підбору препаратів. Досліджено сучасні напрями використання штучного інтелекту. І сучасні підходи до прогнозування рівня та параметрів ін'єкції анестетичних препаратів.

У другому розділі викладено теоретичні основи методів машинного навчання, що складають фундамент розробленої системи. Проаналізовано архітектури рекурентних нейронних мереж, механізм самоуваги та архітектуру Transformer, а також оптимізований варіант Informer. Розглянуто підходи до оцінки якості моделей із застосуванням регресійних, класифікаційних та клінічно орієнтованих метрик.

У третьому розділі розроблено повну архітектуру інтелектуальної системи визначення препаратів для анестезії. Як основу для досліджень обрано відкриту базу даних VitalDB, що містить багатопараметричні інтраопераційні записи 6388 хірургічних пацієнтів; після фільтрації сформовано узгоджений набір даних із 1454 клінічних випадків. Запропоновано архітектуру за принципом пізньої інтеграції, що поєднує статичний кодувальник на основі багатошарового перцептрона та три альтернативні часові кодувальники (LSTM, Transformer, Informer) з двома незалежними регресійними виходами для прогнозування доз пропофолу та реміфентанілу.

У четвертому розділі реалізовано систему на мові Python з використанням фреймворку PyTorch та проведено порівняльне експериментальне дослідження трьох архітектур. За сукупністю результатів найкращі показники продемонстрував Informer: він забезпечив найвищу точність виявлення

болюсних введень для обох препаратів, найнижчу похибку прогнозування реміфентанілу та найвищу швидкість навчання, що робить його оптимальним вибором для практичного застосування. Взаємодія з системою реалізована через REST API із підтримкою формату JSON.

Основні результати роботи планується апробувати на міжнародній науково-практичній конференції з доповіддю на тему «Застосування архітектури Informer для прогнозування дозування анестетиків на основі інтраопераційних часових рядів».

ПЕРЕЛІК ПОСИЛАНЬ

1. Кравець, О., Клигуненко, О., & Єхалов, В. (2025). Анестезіологічна смерть. Проблема та статистика (літературний огляд). *Pain, Anaesthesia & Intensive Care, 1*(110), 7–16. https://doi.org/10.25284/2519-2078.1(110).2025.325912
2. Кириченко, О. А. (2017). Анестезіологія. *Фармацевтична енциклопедія*. https://www.pharmencyclopedia.com.ua/article/2791/anesteziologiya
3. Gropper, M. A., Eriksson, L. I., Fleisher, L. A., Cohen, N. H., Leslie, K., & Johnson-Akeju, O. (2024). *Miller's anesthesia* (2-volume set e-book). Elsevier.
4. Hendrix, J. M., & Garmon, E. H. (2025, February 11). American Society of Anesthesiologists physical status classification system. *StatPearls*. https://www.ncbi.nlm.nih.gov/books/NBK441940/
5. Siddiqui, B. A., & Kim, P. Y. (2023, January 29). Anesthesia stages. *StatPearls*. https://www.ncbi.nlm.nih.gov/books/NBK557596/
6. Benzoni, T., Agarwal, A., & Cascella, M. (2025, March 22). Procedural sedation. *StatPearls*. https://www.ncbi.nlm.nih.gov/books/NBK551685/
7. American Society of Anesthesiologists. (n.d.). *Know your anesthesia risk factors — special considerations for the elderly, children, and infants*. Made for This Moment. https://madeforthismoment.asahq.org/anesthesia-101/types-of-anesthesia/
8. Jo, Y. Y., & Kwak, H. J. (2019). Sedation strategies for procedures outside the operating room. *Yonsei Medical Journal, 60*(6), 491–499. https://doi.org/10.3349/ymj.2019.60.6.491
9. Гаас, А. І., Коваль, Д. О., & Гаас, О. О. (2018). Регіонарна анестезія: традиції та інновації. *PainMedicine Journal, 3*(2), 39–52. https://painmedicine.org.ua/index.php/pnmdcn/article/download/100/87/183

10. Merhavy, Z., Merhavy, C., & Varkey, T. (2021). Anesthetic drugs: A comprehensive overview for anesthesiologists. *Anaesthesia and Intensive Care, 2*, 42–53. https://doi.org/10.46439/anesthesia.2.012
11. Міністерство охорони здоров'я України. (2017). *Клінічні протоколи: що це і чи є вони в Україні*. https://moz.gov.ua/uk/klinichni-protokoli-scho-ce-i-chi-e-v-ukraini-
12. American Society of Anesthesiologists. (2025). *Standards for basic anesthetic monitoring*. https://www.asahq.org/standards-and-practice-parameters/standards-for-basic-anesthetic-monitoring
13. European Society of Anaesthesiology and Intensive Care. (2026, January 14). *Guidelines*. ESAIC. https://esaic.org/guidelines/
14. Kambale, M., & Jadhav, S. (2024). Applications of artificial intelligence in anesthesia: A systematic review. *Saudi Journal of Anaesthesia, 18*(2), 249–256. https://doi.org/10.4103/sja.sja_955_23
15. Bogoń, A., Górska, M., Ostojska, M., Kałuża, I., Dziuba, G., & Dobosz, M. (2024). Artificial intelligence in anesthesiology – A review. *Journal of Pre-Clinical and Clinical Research*. https://doi.org/10.26444/jpccr/191550
16. Kim, J. H., Kim, H., Jang, J. S., Hwang, S. M., Lim, S. Y., Lee, J. J., & Kwon, Y. S. (2021). Development and validation of a difficult laryngoscopy prediction model using machine learning of neck circumference and thyromental height. *BMC Anesthesiology, 21*(1), 125. https://doi.org/10.1186/s12871-021-01343-4
17. Tavolara, T. E., Gurcan, M. N., Segal, S., & Niazi, M. K. K. (2021). Identification of difficult to intubate patients from frontal face images using an ensemble of deep learning models. *Computers in Biology and Medicine, 136*, 104737. https://doi.org/10.1016/j.compbiomed.2021.104737
18. Hayasaka, T., Kawano, K., Kurihara, K., Suzuki, H., Nakane, M., & Kawamae, K. (2021). Creation of an artificial intelligence model for intubation difficulty classification by deep learning (convolutional neural network) using face

images: An observational study. *Journal of Intensive Care, 9*(1), 38. https://doi.org/10.1186/s40560-021-00551-x

19. Luo, J., Lin, S., Wang, L., Ji, H., Zheng, J., Wang, T., Chen, L., Lin, Z., Liu, Z., & Liufu, N. (2025). Predicting delayed extubation after general anesthesia in postanesthesia care unit patients using machine learning: Model development study. *JMIR Medical Informatics, 13*, e72602. https://doi.org/10.2196/72602
20. Hatib, F., Jian, Z., Buddi, S., Lee, C., Settels, J., Sibert, K., Rinehart, J., & Cannesson, M. (2018). Machine-learning algorithm to predict hypotension based on high-fidelity arterial pressure waveform analysis. *Anesthesiology, 129*(4), 663–674. https://doi.org/10.1097/ALN.0000000000002300
21. Huang, X., He, Q., & Zhou, Y. (2025). Prediction of anesthesia resuscitation after painless gastroenteroscopy based on machine learning. In *Proceedings of the 2024 5th International Conference on Intelligent Medicine and Health* (pp. 27–31). https://doi.org/10.1145/3715931.3715936
22. Wang, L., Weng, Y., & Yu, W. (2025). Anesthesia depth prediction from drug infusion history using hybrid AI. *BMC Medical Informatics and Decision Making, 25*. https://doi.org/10.1186/s12911-025-02986-w
23. Hu, Z., Pan, G., Wang, X., & Li, K. (2024). Intelligent algorithm based on deep learning to predict the dosage for anesthesia: A study on prediction of drug efficacy based on deep learning. *Health Science Reports, 7*(5), e2113. https://doi.org/10.1002/hsr2.2113
24. Ren, W., Chen, J., Liu, J., Fu, Z., Yao, Y., Chen, X., & Teng, L. (2022). Feasibility of intelligent drug control in the maintenance phase of general anesthesia based on convolutional neural network. *Heliyon, 9*(1), e12481. https://doi.org/10.1016/j.heliyon.2022.e12481
25. Miyaguchi, N., Takeuchi, K., Kashima, H., Morita, M., & Morimatsu, H. (2021). Predicting anesthetic infusion events using machine learning. *Scientific Reports, 11*(1), 23648. https://doi.org/10.1038/s41598-021-03112-2

26. Chen, M., He, Y., & Yang, Z. (2023). A deep learning framework for anesthesia depth prediction from drug infusion history. *Sensors, 23*(21), 8994. https://doi.org/10.3390/s23218994
27. Zhang, D., & Wang, F. (2025). An interpretable deep actor–critic framework for automated propofol dosing during general anesthesia. *IEEE Access, 13*, 157175–157190. https://doi.org/10.1109/ACCESS.2025.3605643
28. Jamali, N., Sadegheih, A., Lotfi, M. M., & Razavi, H. (2021). Adaptive neuro-fuzzy inference system estimation propofol dose in induction phase during anesthesia: A case study. *International Journal of Engineering Transactions C: Aspects, 34*, 184–191. https://doi.org/10.5829/ije.2021.34.09c.12
29. Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems, 2*(4), 303–314. https://doi.org/10.1007/BF02551274
30. Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2018). Focal loss for dense object detection. *arXiv*. http://arxiv.org/abs/1708.02002
31. Gokcesu, K., & Gokcesu, H. (2021). Generalized Huber loss for robust learning and its efficient minimization for robust statistics. *arXiv*. http://arxiv.org/abs/2108.12627
32. Kingma, D. P., & Ba, J. (2017). Adam: A method for stochastic optimization. *arXiv*. http://arxiv.org/abs/1412.6980
33. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research, 15*(56), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html
34. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
35. Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv*. http://arxiv.org/abs/1412.3555

36. Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., & Hovy, E. (2016). Hierarchical attention networks for document classification. In K. Knight, A. Nenkova, & O. Rambow (Eds.), *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 1480–1489). https://doi.org/10.18653/v1/N16-1174

37. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., & Polosukhin, I. (2023). Attention is all you need. *arXiv*. http://arxiv.org/abs/1706.03762

38. Irani, H., & Metsis, V. (2025). Positional encoding in transformer-based time series models: A survey. *arXiv*. http://arxiv.org/abs/2502.12370

39. Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., & Zhang, W. (2021). Informer: Beyond efficient transformer for long sequence time-series forecasting. *arXiv*. http://arxiv.org/abs/2012.07436

40. Lee, H. C., Park, Y., Yoon, S. B., Yang, S. M., Park, D., & Jung, C. W. (2022). VitalDB, a high-fidelity multi-parameter vital signs database in surgical patients. *Scientific Data*, *9*(1), 279. https://doi.org/10.1038/s41597-022-01411-5

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
!pip install torch>=2.1.0
!pip install numpy>=1.24
!pip install pandas>=2.0
!pip install pyarrow>=12.0
!pip install scikit-learn>=1.3

from google.colab import drive
drive.mount('/content/drive', force_remount=True)

import os
os.environ["PYTORCH_ALLOC_CONF"] = "expandable_segments:True"

"""# Start"""

import torch
import torch.nn as nn
import torch.nn.functional as F

class LSTMEncoder(nn.Module):

    def __init__(
        self,
        input_dim: int,
        hidden_size: int = 128,
        num_layers: int = 2,
        dropout: float = 0.2,
        bidirectional: bool = False,
    ):
        super().__init__()
        self.bidirectional = bidirectional
        self.num_directions = 2 if bidirectional else 1

        self.lstm = nn.LSTM(
```

```
            input_size=input_dim,
            hidden_size=hidden_size,
            num_layers=num_layers,
            batch_first=True,
            dropout=dropout if num_layers > 1 else 0.0,
            bidirectional=bidirectional,
        )
        # Project back to hidden_size regardless of bidirectionality
        self.proj = nn.Linear(hidden_size * self.num_directions, hidden_size)
        self.norm = nn.LayerNorm(hidden_size)

    def forward(self, x: torch.Tensor, padding_mask=None) -> torch.Tensor:
        # x: (B, T, D)
        _, (h_n, _) = self.lstm(x)
        # h_n: (num_layers * num_directions, B, hidden_size)
        # Take last layer, concatenate directions
        if self.bidirectional:
            h = torch.cat([h_n[-2], h_n[-1]], dim=-1)   # (B, 2*H)
        else:
            h = h_n[-1]                                 # (B, H)
        return self.norm(self.proj(h))                  # (B, H)


class AttentionLSTMEncoder(nn.Module):
    def __init__(
        self,
        input_dim: int,
        hidden_size: int = 128,
        num_layers: int = 2,
        dropout: float = 0.2,
        bidirectional: bool = False,
    ):
        super().__init__()
        self.num_directions = 2 if bidirectional else 1

        self.lstm = nn.LSTM(
            input_size=input_dim,
```

```
            hidden_size=hidden_size,
            num_layers=num_layers,
            batch_first=True,
            dropout=dropout if num_layers > 1 else 0.0,
            bidirectional=bidirectional,
        )
        attn_dim = hidden_size * self.num_directions
        self.attn = nn.Linear(attn_dim, 1)
        self.proj = nn.Linear(attn_dim, hidden_size)
        self.norm = nn.LayerNorm(hidden_size)

    def forward(self, x: torch.Tensor, padding_mask=None) -> torch.Tensor:
        # x: (B, T, D)
        out, _ = self.lstm(x)          # (B, T, H * num_dir)
        scores = self.attn(out)        # (B, T, 1)
        if padding_mask is not None:
          scores[padding_mask.unsqueeze(-1)] = -1e9
        weights = torch.softmax(scores, dim=1)
        context = (weights * out).sum(dim=1)   # (B, H * num_dir)
        return self.norm(self.proj(context))   # (B, H)


class TransformerTSEncoder(nn.Module):
    def __init__(
        self,
        input_dim  : int,
        hidden_size: int   = 256,
        num_layers : int   = 4,
        nhead      : int   = 8,
        dropout    : float = 0.1,
        max_len    : int   = 512,
    ):
        super().__init__()
        assert hidden_size % nhead == 0, "hidden_size must be divisible by nhead"

        self.input_proj = nn.Linear(input_dim, hidden_size)
```

```
        self.pool_attn = nn.Linear(hidden_size, 1)

        self.pos_emb = nn.Embedding(max_len, hidden_size)

        encoder_layer = nn.TransformerEncoderLayer(
            d_model         = hidden_size,
            nhead           = nhead,
            dim_feedforward = hidden_size * 2,
            dropout         = dropout,
            batch_first     = True,
            norm_first      = True,
        )
        self.encoder = nn.TransformerEncoder(encoder_layer, num_layers=num_layers)

        self.out_proj = nn.Linear(hidden_size, hidden_size)
        self.norm     = nn.LayerNorm(hidden_size)

    def forward(self, x: torch.Tensor, padding_mask=None) -> torch.Tensor:
        B, T, _ = x.shape
        positions = torch.arange(T, device=x.device).unsqueeze(0)  # (1, T)

        h = self.input_proj(x) + self.pos_emb(positions)  # (B, T, H)

        h = self.encoder(h, src_key_padding_mask=padding_mask)  # (B, T, H)

        if padding_mask is not None:
            valid_mask = (~padding_mask).unsqueeze(-1)  # (B, T, 1)
        else:
            valid_mask = torch.ones_like(h[..., :1])

        scores = self.pool_attn(h)  # (B, T, 1)
        scores = scores.masked_fill(~valid_mask, float('-inf'))

        weights = torch.softmax(scores, dim=1)  # (B, T, 1)

        h = torch.sum(h * weights, dim=1)  # (B, H)
```

```
        return self.norm(self.out_proj(h))  # (B, H)

class ProbSparseAttention(nn.Module):
    def __init__(
        self,
        d_model : int,
        nhead   : int,
        factor  : int   = 5,
        dropout : float = 0.1,
    ):
        super().__init__()
        assert d_model % nhead == 0, "d_model must be divisible by nhead"
        self.nhead   = nhead
        self.d_k     = d_model // nhead
        self.factor  = factor
        self.dropout = nn.Dropout(dropout)

        self.q_proj = nn.Linear(d_model, d_model)
        self.k_proj = nn.Linear(d_model, d_model)
        self.v_proj = nn.Linear(d_model, d_model)
        self.out    = nn.Linear(d_model, d_model)

    def _prob_qk(
        self,
        Q       : torch.Tensor,   # (B, H, T, d_k)
        K       : torch.Tensor,   # (B, H, T, d_k)
        sample_k: int,
        n_top   : int,
    ) -> torch.Tensor:
        B, H, T, d = Q.shape

        idx_keys = torch.randint(T, (B, H, sample_k), device=Q.device)
        idx_exp  = idx_keys.unsqueeze(-1).expand(B, H, sample_k, d)
        K_sample = K.gather(2, idx_exp)

        Q_K = torch.einsum("bhid,bhjd->bhij", Q, K_sample) / (d ** 0.5)
        M = Q_K.max(-1).values - Q_K.mean(-1)
```

```
    top_idx = M.topk(n_top, dim=-1).indices   # (B, H, n_top)
    return top_idx

def forward(
    self,
    x          : torch.Tensor,         # (B, T, d_model)
    padding_mask: torch.Tensor = None,   # (B, T) bool,
) -> torch.Tensor:
    B, T, _ = x.shape
    H, d = self.nhead, self.d_k

    # Project and reshape to (B, H, T, d_k)
    Q = self.q_proj(x).view(B, T, H, d).transpose(1, 2)
    K = self.k_proj(x).view(B, T, H, d).transpose(1, 2)
    V = self.v_proj(x).view(B, T, H, d).transpose(1, 2)

    # ProbSparse: select top queries
    sample_k = min(int(self.factor * math.log(T + 1)), T)
    n_top    = min(sample_k, T)

    top_idx = self._prob_qk(Q, K, sample_k, n_top)   # (B, H, n_top)

    # Gather the selected query vectors: (B, H, n_top, d_k)
    idx_exp = top_idx.unsqueeze(-1).expand(B, H, n_top, d)
    Q_top   = Q.gather(2, idx_exp)

    # Full attention for the selected queries only
    # scores: (B, H, n_top, T)
    scores = torch.einsum("bhqd,bhkd->bhqk", Q_top, K) / (d ** 0.5)

    if padding_mask is not None:
        # padding_mask: (B, T)
        # broadcast to (B, 1, 1, T)
        scores = scores.masked_fill(
            padding_mask.unsqueeze(1).unsqueeze(2), -1e9
        )
```

```
        attn = torch.softmax(scores, dim=-1)   # (B, H, n_top, T)
        attn = self.dropout(attn)

        # (B, H, n_top, d_k)
        attended = torch.einsum("bhqt,bhtd->bhqd", attn, V)

        # out: (B, H, T, d_k) - initialise all positions with temporal mean of V
        out = V.mean(dim=2, keepdim=True).expand(B, H, T, d).clone()

        # Scatter attended values back into their original query positions
        top_exp = top_idx.unsqueeze(-1).expand(B, H, n_top, d)
        out.scatter_(2, top_exp, attended)

        # Reshape and project
        out = out.transpose(1, 2).reshape(B, T, H * d)   # (B, T, d_model)
        return self.out(out)


#
_____________________________________________________________________________
_____________________________________________


class InformerTSEncoder(nn.Module):

    def __init__(
        self,
        input_dim  : int,
        hidden_size: int   = 256,
        num_layers : int   = 3,
        nhead      : int   = 8,
        factor     : int   = 5,
        dropout    : float = 0.1,
        max_len    : int   = 512,
    ):
        super().__init__()
        assert hidden_size % nhead == 0, "hidden_size must be divisible by nhead"
```

```
        self.input_proj = nn.Linear(input_dim, hidden_size)
        self.pos_emb    = nn.Embedding(max_len, hidden_size)

        self.attn_layers = nn.ModuleList([
            ProbSparseAttention(hidden_size, nhead, factor, dropout)
            for _ in range(num_layers)
        ])

        # One ConvDistil block between consecutive attention layers
        self.distil_convs = nn.ModuleList([
            nn.Sequential(
                nn.Conv1d(hidden_size, hidden_size, kernel_size=3, padding=1),
                nn.ELU(),
                nn.MaxPool1d(kernel_size=2, stride=2),
            )
            for _ in range(num_layers - 1)
        ])

        self.norms    = nn.ModuleList([nn.LayerNorm(hidden_size) for _ in
range(num_layers)])
        self.drop     = nn.Dropout(dropout)
        self.norm_out = nn.LayerNorm(hidden_size)

    def _inject_pos(self, h: torch.Tensor) -> torch.Tensor:
        T   = h.shape[1]
        pos = torch.arange(T, device=h.device).unsqueeze(0)   # (1, T)
        return h + self.pos_emb(pos)

    def _downsample_mask(
        self,
        padding_mask: torch.Tensor,   # (B, T_in) bool
        T_new       : int,
    ) -> torch.Tensor:
        T_in = padding_mask.shape[1]
        # Trim to the largest even length ≤ T_in that maps to T_new pairs
        T_trim = T_new * 2
```

```
    if T_in < T_trim:
        # Pad the mask with True (padding) on the right to reach T_trim
        pad = padding_mask.new_ones(padding_mask.shape[0], T_trim - T_in)
        padding_mask = torch.cat([padding_mask, pad], dim=1)
    else:
        padding_mask = padding_mask[:, :T_trim]

    # (B, T_new, 2) → OR across the pair → (B, T_new)
    return padding_mask.view(padding_mask.shape[0], T_new, 2).any(dim=-1)

def forward(
    self,
    x           : torch.Tensor,         # (B, T, input_dim)
    padding_mask: torch.Tensor = None,   # (B, T) bool, True = padding
) -> torch.Tensor:

    h = self.input_proj(x)          # (B, T, H)
    h = self._inject_pos(h)         # add positional info for the original T

    for i, attn in enumerate(self.attn_layers):
        h = h + self.drop(attn(h, padding_mask))
        h = self.norms[i](h)

        if i < len(self.distil_convs):
            # (B, T, H) → conv expects (B, H, T) → back to (B, T/2, H)
            h = self.distil_convs[i](h.transpose(1, 2)).transpose(1, 2)

            T_new = h.shape[1]

            if padding_mask is not None:
                padding_mask = self._downsample_mask(padding_mask, T_new)

            h = self._inject_pos(h)

    if padding_mask is not None:
        real_mask = (~padding_mask).float().unsqueeze(-1)   # (B, T, 1)
        h = (h * real_mask).sum(1) / real_mask.sum(1).clamp(min=1)
```

```
        else:
            h = h.mean(dim=1)

        return self.norm_out(h)   # (B, H)

def _build_ts_encoder(arch: str, input_dim: int, hidden_size: int, num_layers: int,
dropout: float):
    if arch == "lstm":
        return AttentionLSTMEncoder(input_dim, hidden_size, num_layers, dropout,
True)
    elif arch == "informer":
        nhead = 8 if hidden_size >= 256 else 4
        return InformerTSEncoder(input_dim, hidden_size, num_layers, nhead,
dropout=dropout)
    else:
        raise ValueError(f"Unknown arch: {arch!r}. Choose 'lstm', 'transformer', or
'informer'.")

class GatedStaticEncoder(nn.Module):
    def __init__(self, input_dim: int, hidden_size: int = 128,
                 dropout: float = 0.2):
        super().__init__()

        self.input_proj = nn.Sequential(
            nn.Linear(input_dim, hidden_size),
            nn.LayerNorm(hidden_size),
        )

        self.block = nn.Sequential(
            nn.Linear(hidden_size, hidden_size),
            nn.GELU(),
            nn.Dropout(dropout),
            nn.Linear(hidden_size, hidden_size),
            nn.LayerNorm(hidden_size),
        )

        self.gate = nn.Sequential(
```

```
            nn.Linear(hidden_size, hidden_size),
            nn.Sigmoid(),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        h = self.input_proj(x)
        h = h + self.block(h)
        return self.gate(h) * h


# ──────────────────────────────────────────────────────────────────────
# Improved: gated fusion
# ──────────────────────────────────────────────────────────────────────

class GatedFusion(nn.Module):

    def __init__(self, hidden_size: int = 128, dropout: float = 0.2):
        super().__init__()

        # Static context gates the ts stream
        self.context_gate = nn.Sequential(
            nn.Linear(hidden_size, hidden_size),
            nn.Sigmoid(),
        )

        # Fuse gated ts + static
        self.fusion = nn.Sequential(
            nn.Linear(hidden_size * 2, hidden_size),
            nn.GELU(),
            nn.Dropout(dropout),
            nn.LayerNorm(hidden_size),
        )

    def forward(self, ts_repr: torch.Tensor,
                static_repr: torch.Tensor) -> torch.Tensor:
        gate = self.context_gate(static_repr)       # (B, H)
        gated_ts = gate * ts_repr                   # (B, H)
```

```
        fused = torch.cat([gated_ts, static_repr], dim=-1)
        return self.fusion(fused)                  # (B, H)


# ──────────────────────────────────────────────────────────────────
# Improved: shared drug-context trunk
# ──────────────────────────────────────────────────────────────────

class DrugContextTrunk(nn.Module):

    def __init__(self, hidden_size: int = 128, dropout: float = 0.2):
        super().__init__()
        self.block1 = nn.Sequential(
            nn.Linear(hidden_size, hidden_size * 2),
            nn.GELU(),
            nn.Dropout(dropout),
            nn.Linear(hidden_size * 2, hidden_size),
            nn.LayerNorm(hidden_size),
        )
        self.block2 = nn.Sequential(
            nn.Linear(hidden_size, hidden_size),
            nn.GELU(),
            nn.Dropout(dropout),
            nn.LayerNorm(hidden_size),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = x + self.block1(x)
        x = x + self.block2(x)
        return x

class DrugPredictionHead(nn.Module):
    """
    Input : (B, H)
    Output: (B,)  - non-negative scalar per sample
    """
```

```
    def __init__(self, hidden_size: int = 128, dropout: float = 0.2):
        super().__init__()
        mid = hidden_size // 2
        self.net = nn.Sequential(
            nn.Linear(hidden_size, mid * 2),
            nn.GELU(),
            nn.Dropout(dropout),
            nn.Linear(mid * 2, mid),
            nn.GELU(),
            nn.Linear(mid, 1),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        raw = self.net(x).squeeze(-1)       # (B,)
        return raw

class AnesthesiaDrugPredictor(nn.Module):
    def __init__(
        self,
        ts_input_dim: int,
        static_input_dim: int,
        hidden_size: int = 128,
        num_layers: int = 2,
        dropout: float = 0.2,
        arch : str = "lstm",
        bidirectional: bool = False,
    ):
        super().__init__()

        self.ts_encoder     = _build_ts_encoder(arch, ts_input_dim, hidden_size,
num_layers, dropout)
        self.static_encoder = GatedStaticEncoder(
            input_dim=static_input_dim,
            hidden_size=hidden_size,
            dropout=dropout,
        )
        self.fusion = GatedFusion(hidden_size=hidden_size, dropout=dropout)
```

```
        self.trunk = DrugContextTrunk(hidden_size=hidden_size, dropout=dropout)

        self.ppf20_head  = DrugPredictionHead(hidden_size, dropout)
        self.rftn20_head = DrugPredictionHead(hidden_size, dropout)

    def forward(
        self,
        ts: torch.Tensor,
        static: torch.Tensor,
        mask: torch.Tensor = None,
    ) -> tuple[torch.Tensor, torch.Tensor]:
        ts_repr     = self.ts_encoder(ts, mask)
        static_repr = self.static_encoder(static)

        fused = self.fusion(ts_repr, static_repr)
        context = self.trunk(fused)

        return self.ppf20_head(context), self.rftn20_head(context)

def weighted_mse(pred, target, spike_threshold=3.0, spike_weight=10.0):
    residuals = (pred - target) ** 2
    weights   = torch.where(
        target > spike_threshold,
        torch.full_like(target, spike_weight),
        torch.ones_like(target)
    )
    return (weights * residuals).mean()

def weighted_mse_np(pred, target, spike_threshold=3.0, spike_weight=10.0):
    residuals = (pred - target) ** 2
    weights = np.where(target > spike_threshold, spike_weight, 1.0)
    return np.mean(weights * residuals)

"""
Dataset & DataLoader utilities
"""
```

```
import numpy as np
import pandas as pd
import torch
from torch.utils.data import Dataset, DataLoader, Subset
from typing import Optional
import warnings

warnings.filterwarnings("ignore")

TS_FEATURES = [
    'PPF20_CUM_NORM',
    'RFTN20_CUM_NORM',
    'TIME_ELAPSED_NORM',
    'PPF20_SINCE_BOLUS_NORM',
    'RFTN20_SINCE_BOLUS_NORM',
    'ART_MBP',
    'PLETH_SPO2',
    'BIS',
    'ETCO2',
    'ART_MBP_DIFF',
    'PLETH_SPO2_DIFF',
    'BIS_DIFF',
    'ETCO2_DIFF',
    'PPF20_RATE',
    'RFTN20_RATE',
    'PPF20_RATE_DIFF',
    'RFTN20_RATE_DIFF'
 ]

TARGET_PPF20  = "PPF20_RATE"
TARGET_RFTN20 = "RFTN20_RATE"

STATIC_EXCLUDE = ["caseid"]

def load_parquets(data_dir: str) -> tuple[pd.DataFrame, pd.DataFrame,
pd.DataFrame, pd.DataFrame, pd.DataFrame]:
    import os
```

```
    d = data_dir.rstrip("/")
    ts       = pd.read_parquet(f"{d}/ts.parquet")
    static   = pd.read_parquet(f"{d}/static.parquet")
    train_ix = pd.read_parquet(f"{d}/train_index.parquet")
    val_ix   = pd.read_parquet(f"{d}/validation_index.parquet")
    test_ix  = pd.read_parquet(f"{d}/test_index.parquet")
    return ts, static, train_ix, val_ix, test_ix


def resolve_ts_features(ts: pd.DataFrame) -> list[str]:
    available = [f for f in TS_FEATURES if f in ts.columns]
    missing   = [f for f in TS_FEATURES if f not in ts.columns]
    if missing:
        print(f"[WARNING] Missing TS features (will be skipped): {missing}")
    return available


def resolve_static_features(static: pd.DataFrame) -> list[str]:
    cols = [c for c in static.columns if c not in STATIC_EXCLUDE]
    # Keep only numeric
    cols = [c for c in cols if pd.api.types.is_numeric_dtype(static[c])]
    return cols

CACHE_DIR = "/content/drive/MyDrive/Mag2026Data/data/window_cache"
def _cache_path(split: str, window_size: int, horizon: int, target_type : str) -> str:
    target_type_name = target_type if not target_type == "mean" else ""
    return os.path.join(CACHE_DIR,
f"{split}_w{window_size}_h{horizon}{target_type_name}.npz")

def _save_cache(path: str, ts, static, ppf, rftn, mask):
    os.makedirs(os.path.dirname(path), exist_ok=True)
    np.savez_compressed(path, ts=ts, static=static, ppf=ppf, rftn=rftn, mask=mask)
    size_mb = os.path.getsize(path) / 1e6
    print(f"Cache saved → {path}  ({size_mb:.0f} MB)")

def _load_cache(path: str):
    print(f"Loading cache from Drive … {path}")
```

```
    data = np.load(path)
    print(f"  ts={data['ts'].shape}  static={data['static'].shape}  - done.")
    return data["ts"], data["static"], data["ppf"], data["rftn"], data["mask"]

def apply_target_type(datarow, horizon, target_type):
    if target_type == "mean":
        return float(datarow.mean())

    elif target_type == "max":
        return float(datarow.max())

    elif target_type == "closeness_bias":
        sigma = max(horizon / 2, 1e-6)

        x = np.arange(horizon)

        weights = np.exp(-(x ** 2) / (2 * sigma ** 2))

        weights = weights / weights.sum()

        return float((weights * datarow).sum())

    elif target_type == "next":
        return float(datarow[0])
    else:
        raise ValueError(
            f"Unknown target_type: {target_type}. "
        )



def _build_windows_raw(
    index_df       : pd.DataFrame,
    ts_df          : pd.DataFrame,
    static_df      : pd.DataFrame,
    ts_features    : list[str],
    static_features: list[str],
```

```
    window_size    : int,
    horizon        : int,
    target_type    : str,
    desc           : str,
) -> tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray]:
    """
    Pre-slice every (caseid, t) pair in index_df into a fixed-size numpy window
    Returns
    ts_arr     : float32  (N, window_size, ts_features)
    static_arr : float32  (N, static_features)
    ppf_arr    : float32  (N,)
    rftn_arr   : float32  (N,)
    """
    n           = len(index_df)
    n_ts        = len(ts_features)
    n_static    = len(static_features)
    target_ppf  = "PPF20_RATE"
    target_rftn = "RFTN20_RATE"

    mask_out   = np.zeros((n, window_size),         dtype=bool)
    ts_out     = np.zeros((n, window_size, n_ts),   dtype=np.float32)
    static_out = np.zeros((n, n_static),            dtype=np.float32)
    ppf_out    = np.zeros(n,                        dtype=np.float32)
    rftn_out   = np.zeros(n,                        dtype=np.float32)

    ts_by_case: dict[int, tuple[np.ndarray, np.ndarray]] = {}
    for caseid, grp in ts_df.groupby("caseid"):
        grp   = grp.sort_values("time").reset_index(drop=True)
        times = grp["time"].to_numpy(dtype=np.int32)
        vals  = grp[ts_features].ffill().fillna(0).to_numpy(dtype=np.float32)
        ts_by_case[int(caseid)] = (times, vals)

    static_indexed = static_df.set_index("caseid") if "caseid" in static_df.columns else
static_df
    static_vals_map = {
        int(cid): static_indexed.loc[cid,
static_features].fillna(0).to_numpy(dtype=np.float32)
```

```
    for cid in static_indexed.index
    if cid in ts_by_case
}

print(f"{desc}: {n:,} samples …")
report_every = max(1, n // 20)

index_arr = index_df.reset_index(drop=True)

for i in range(n):
    if i % report_every == 0:
        print(f"  {i / n * 100:5.1f}%")

    row    = index_arr.iloc[i]
    caseid = int(row["caseid"])
    t      = int(row["t"])

    times, vals = ts_by_case[caseid]

    t_start   = t - window_size
    mask_in   = (times >= t_start) & (times < t)
    window    = vals[mask_in]                    # (<=W, D)

    actual_len = len(window)
    if actual_len >= window_size:
        ts_out[i] = window[-window_size:]
    elif actual_len < window_size:
        mask_out[i, : window_size - actual_len] = True
        ts_out[i, window_size - actual_len:] = window

    t_end       = t + horizon
    mask_future = (times > t) & (times <= t_end)
    future_vals = vals[mask_future]                # (<=horizon, D)

    ppf_col  = ts_features.index(target_ppf)
    rftn_col = ts_features.index(target_rftn)
```

```
        ppf_out[i]  = apply_target_type(future_vals[:, ppf_col], horizon, target_type)
        rftn_out[i] = apply_target_type(future_vals[:, rftn_col], horizon, target_type)

        static_out[i] = static_vals_map.get(
            caseid, np.zeros(n_static, dtype=np.float32)
        )

    print(f"  100.0% - done ({n:,} windows built)")
    return ts_out, mask_out, static_out, ppf_out, rftn_out

def _build_windows(
    index_df        : pd.DataFrame,
    ts_df           : pd.DataFrame,
    static_df       : pd.DataFrame,
    ts_features     : list[str],
    static_features : list[str],
    window_size     : int,
    horizon         : int,
    target_type     : str,
    desc            : str = "Building windows",
    split           : str = "unknown",      # "train", "val", "test"
    use_cache       : bool = True,
):
    cache_path = _cache_path(split, window_size, horizon, target_type)

    if use_cache and os.path.exists(cache_path):
        return _load_cache(cache_path)

    ts, static, ppf, rftn, mask = _build_windows_raw(
        index_df, ts_df, static_df,
        ts_features, static_features,
        window_size, horizon, target_type, desc,
    )

    if use_cache:
        _save_cache(cache_path, ts, static, ppf, rftn, mask)
```

```
    return ts, static, ppf, rftn, mask

class PreloadedTrainDataset(Dataset):
    """
    All windows are pre-built into contiguous numpy arrays at init time in RAM.
    """

    def __init__(
        self,
        index_df       : pd.DataFrame,
        ts_df          : pd.DataFrame,
        static_df      : pd.DataFrame,
        ts_features    : list[str],
        static_features: list[str],
        window_size    : int = 300,
        horizon        : int = 6,
        target_type    : str = "mean",
        split          : str = "train",
        use_cache      : bool= True
    ):
        self.ts_arr, self.mask_arr, self.static_arr, self.ppf_arr, self.rftn_arr =
_build_windows(
            index_df, ts_df, static_df,
            ts_features, static_features,
            window_size, horizon, target_type,
            desc="PreloadedTrainDataset",
            split = split, use_cache=use_cache
        )

    def __len__(self) -> int:
        return len(self.ppf_arr)

    def __getitem__(self, idx: int) -> dict:
        return {
            "ts"           : torch.from_numpy(self.ts_arr[idx]),
            "padding_mask"  : torch.from_numpy(self.mask_arr[idx]),
            "static"       : torch.from_numpy(self.static_arr[idx]),
```

```
            "ppf20"         : torch.tensor(self.ppf_arr[idx]),
            "rftn20"        : torch.tensor(self.rftn_arr[idx]),
        }
class GPUValStore:
    """
    All windows are pre-built into contiguous numpy arrays at init time in GPU.
    """

    def __init__(
        self,
        index_df       : pd.DataFrame,
        ts_df          : pd.DataFrame,
        static_df      : pd.DataFrame,
        ts_features    : list[str],
        static_features: list[str],
        window_size    : int = 300,
        horizon        : int = 6,
        device         : str = "cuda",
        split          : str = "train",
        target_type    : str = "mean",
        use_cache      : bool= True,
        batch_size     : int = 2048,
    ):
        self.device     = device
        self.batch_size = batch_size

        ts_np, mask_np, static_np, ppf_np, rftn_np = _build_windows(
            index_df, ts_df, static_df,
            ts_features, static_features,
            window_size, horizon, target_type,
            desc="GPUValStore",
            split = split, use_cache=use_cache
        )

        print("Uploading val tensors to GPU …", end=" ")
        # non_blocking=True overlaps the H→D transfer with any pending CPU work
        self.ts     = torch.from_numpy(ts_np).to(device, non_blocking=True)
```

```
    self.mask   = torch.from_numpy(mask_np).to(device, non_blocking=True)
    self.static = torch.from_numpy(static_np).to(device, non_blocking=True)
    self.ppf20  = torch.from_numpy(ppf_np).to(device, non_blocking=True)
    self.rftn20 = torch.from_numpy(rftn_np).to(device, non_blocking=True)
    torch.cuda.synchronize()
    print("done.")

    gb = (self.ts.nbytes + self.static.nbytes +
          self.ppf20.nbytes + self.rftn20.nbytes) / 1e9
    print(f"Val store on GPU: {gb:.2f} GB")

def __len__(self) -> int:
    return len(self.ppf20)

def evaluate(self, model: torch.nn.Module, cfg) -> dict:
    model.eval()
    n = len(self.ppf20)

    all_ppf_pred  = []
    all_rftn_pred = []

    with torch.no_grad():
        for start in range(0, n, self.batch_size):
            end = min(start + self.batch_size, n)

            ts_b     = self.ts    [start:end]   # already on GPU
            static_b = self.static[start:end]
            mask_b   = self.mask  [start:end]

            ppf_pred, rftn_pred = model(ts_b, static_b, mask_b)

            all_ppf_pred .append(ppf_pred)
            all_rftn_pred.append(rftn_pred)

    ppf_pred_all  = torch.cat(all_ppf_pred ).cpu().numpy()
    rftn_pred_all = torch.cat(all_rftn_pred).cpu().numpy()
    ppf_tgt       = self.ppf20 .cpu().numpy()
```

```
        rftn_tgt      = self.rftn20.cpu().numpy()

        metrics = {"loss": float(
            0.5 * weighted_mse_np(ppf_pred_all, ppf_tgt) +
            0.5 * weighted_mse_np(rftn_pred_all, rftn_tgt)
        )}
        metrics.update(compute_metrics(ppf_pred_all,  ppf_tgt,  prefix="ppf20"))
        metrics.update(compute_metrics(rftn_pred_all, rftn_tgt, prefix="rftn20"))
        return metrics

class AnesthesiaDataset(Dataset):

    def __init__(
        self,
        index_df: pd.DataFrame,
        ts_df: pd.DataFrame,
        static_df: pd.DataFrame,
        ts_features: list[str],
        static_features: list[str],
        window_size: int = 60,
        horizon: int = 6,
    ):
        self.index_df       = index_df.reset_index(drop=True)
        self.ts_features    = ts_features
        self.static_features = static_features
        self.window_size    = window_size
        self.horizon        = horizon

        self.ts_by_case: dict[int, pd.DataFrame] = {}
        for caseid, grp in ts_df.groupby("caseid"):
            grp = grp.sort_values("time").reset_index(drop=True)
            self.ts_by_case[caseid] = grp

        self.static_df = static_df.set_index("caseid") if "caseid" in static_df.columns else
static_df

        static_vals = self.static_df[static_features].fillna(0).values.astype(np.float32)
```

```
        self.static_scaled = {
            cid: static_vals[i]
            for i, cid in enumerate(self.static_df.index)
        }

    def __len__(self) -> int:
        return len(self.index_df)

    def __getitem__(self, idx: int) -> dict:
        row    = self.index_df.iloc[idx]
        caseid = int(row["caseid"])
        t      = int(row["t"])

        ts_case = self.ts_by_case[caseid]

        t_start = t - self.window_size
        window  = ts_case[
            (ts_case["time"] >= t_start) & (ts_case["time"] < t)
        ][self.ts_features].fillna(method="ffill").fillna(0)

        mask = None

        if len(window) < self.window_size:
            mask = np.array(
                [True] * (self.window_size - len(window)) +
                [False] * len(window),
                dtype=bool
            )
            pad = pd.DataFrame(
                np.zeros((self.window_size - len(window), len(self.ts_features))),
                columns=self.ts_features,
            )
            window = pd.concat([pad, window], ignore_index=True)
        else:
            mask = np.array([False] * (len(window)), dtype=bool)
```

```
        ts_arr = window.values[-self.window_size:].astype(np.float32)  # (T, D)

        t_end   = t + self.horizon
        future  = ts_case[
            (ts_case["time"] > t) & (ts_case["time"] <= t_end)
        ]

        ppf20_target  = float(future[TARGET_PPF20].mean())  if len(future) > 0 else
0.0
        rftn20_target = float(future[TARGET_RFTN20].mean()) if len(future) > 0 else
0.0

        static_arr = self.static_scaled.get(caseid, np.zeros(len(self.static_features),
dtype=np.float32))

        return {
            "ts":     torch.tensor(ts_arr, dtype=torch.float32),
            "padding_mask": mask,
            "static": torch.tensor(static_arr, dtype=torch.float32),
            "ppf20":  torch.tensor(ppf20_target, dtype=torch.float32),
            "rftn20": torch.tensor(rftn20_target, dtype=torch.float32),
            "caseid": caseid,
            "t":      t,
        }

def auroc(probs: ===, labels: np.ndarray):
    n_pos = labels.sum()
    n_neg = len(labels) - n_pos
    if n_pos == 0 or n_neg == 0:
        return float("nan")
    order  = np.argsort(-probs)
    labels = labels[order]
    tpr, fpr = [0.0], [0.0]
    tp = fp = 0
    for l in labels:
        if l:
            tp += 1
```

```
            else:
                fp += 1
            tpr.append(tp / n_pos)
            fpr.append(fp / n_neg)
        tpr.append(1.0); fpr.append(1.0)
        return float(np.trapz(tpr, fpr))

    def compute_metrics(preds: np.ndarray, targets: np.ndarray, prefix: str) -> dict:
        mae  = np.mean(np.abs(preds - targets))
        rmse = np.sqrt(np.mean((preds - targets) ** 2))
        ss_res = np.sum((targets - preds) ** 2)
        ss_tot = np.sum((targets - targets.mean()) ** 2)
        r2 = 1 - ss_res / (ss_tot + 1e-8)
        bias  = np.mean((preds - targets))

        out = {
            f"{prefix}_mae":  float(mae),
            f"{prefix}_rmse": float(rmse),
            f"{prefix}_r2":   float(r2),
            f"{prefix}_bias": float(bias),
        }

        spike_threshold = 3.0
        spike_mask = targets > spike_threshold
        spike_count = int(spike_mask.sum())
        out[f"{prefix}_spike_count"] = spike_count

        if spike_count > 0:
            sp_tgt  = targets[spike_mask]
            sp_pred = preds[spike_mask]

            out[f"{prefix}_spike_mae"]  = float(np.mean(np.abs(sp_pred - sp_tgt)))
            out[f"{prefix}_spike_rmse"] = float(np.sqrt(np.mean((sp_pred - sp_tgt) ** 2)))
            out[f"{prefix}_spike_bias"] = float(np.mean(sp_pred - sp_tgt))

            pred_spike = preds > spike_threshold
```

```
            tp = int((pred_spike & spike_mask).sum())
            fp = int((pred_spike & ~spike_mask).sum())
            fn = int((~pred_spike & spike_mask).sum())

            recall    = tp / (tp + fn + 1e-8)
            precision = tp / (tp + fp + 1e-8)

            auroc = auroc(pred_spike,sp_tgt)

            out[f"{prefix}_spike_recall"]    = float(recall)
            out[f"{prefix}_spike_precision"] = float(precision)
            out[f"{prefix}_auroc"] = float(auroc)
        else:
            for key in ("spike_mae", "spike_rmse", "spike_bias",
                        "spike_recall", "spike_precision"):
                out[f"{prefix}_{key}"] = float("nan")

        return out

    def build_dataloaders(
        data_dir: str,
        batch_size: int = 64,
        window_size: int = 60,
        horizon: int = 6,
        target_type : str = "mean",
        num_workers: int = 4,
        device : str = "cpu",
        val_batch_size : int = 2048
    ) -> tuple[DataLoader, DataLoader, DataLoader, list[str], list[str]]:
        ts, static, train_ix, val_ix, test_ix = load_parquets(data_dir)

        ts_features     = resolve_ts_features(ts)
        static_features = resolve_static_features(static)

        print(f"TS features     : {len(ts_features)}")
        print(f"Static features : {len(static_features)}")
```

```
    common_kw = dict(
        ts_df=ts,
        static_df=static,
        ts_features=ts_features,
        static_features=static_features,
        window_size=window_size,
        horizon=horizon,
        target_type=target_type,
    )

    print("\n── Building train dataset ──────────────────────────────────────")
    train_ds = PreloadedTrainDataset(train_ix, split="train", **common_kw)

    print("\n── Building val GPU store ──────────────────────────────────────")
    val_store = GPUValStore(
        val_ix, **common_kw,
        split      = "val",
        device     = device,
        batch_size = val_batch_size
    )

    test_ds     = PreloadedTrainDataset(test_ix, split="test", **common_kw)
    test_loader = DataLoader(
        test_ds,
        batch_size  = batch_size,
        shuffle     = False,
        num_workers = num_workers,
        pin_memory  = True,
    )

    print(f"Train samples: {len(train_ds):,} | Val: {len(val_store):,} | Test: {len(test_ds):,}")
    return train_ds, val_store, test_loader, ts_features, static_features

import os
```

```
import math
import time
import json
from dataclasses import dataclass, field, asdict

import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from torch.optim import AdamW
from torch.optim.lr_scheduler import CosineAnnealingWarmRestarts
import shutil

@dataclass
class TrainConfig:
    # Data
    data_dir    : str   = "./data"
    batch_size  : int   = 64
    window_size : int   = 60
    horizon     : int   = 6
    num_workers : int   = 4

    # Model
    arch          : str   = "lstm"
    hidden_size   : int   = 128
    num_layers    : int   = 3
    dropout       : float = 0.2
    use_attention : bool  = True
    bidirectional : bool  = False

    # Optimisation
    lr          : float = 3e-4
    weight_decay : float = 1e-5
    max_epochs  : int   = 50
    patience    : int   = 7

    # Loss weights
```

```
    ppf20_weight  : float = 0.5
    rftn20_weight : float = 0.5

    # Chunks setting
    chunks_per_epoch : int = 10
    val_every_n_chunks : int = 5

    # Checkpointing
    checkpoint_dir : str  = "./checkpoints"
    run_name       : str  = "run_001"

    # Hardware
    device : str = "cuda" if torch.cuda.is_available() else "cpu"



def make_chunk_loaders(train_dataset, cfg: TrainConfig):
    n = len(train_dataset)
    indices = torch.randperm(n).tolist()

    chunk_size = math.ceil(n / cfg.chunks_per_epoch)
    loaders = []
    for start in range(0, n, chunk_size):
        chunk_indices = indices[start : start + chunk_size]
        subset  = Subset(train_dataset, chunk_indices)
        loader  = DataLoader(
            subset,
            batch_size  = cfg.batch_size,
            shuffle     = True,
            num_workers = cfg.num_workers,
            pin_memory  = True,
        )
        loaders.append(loader)

    return loaders

def run_chunk(
```

```
    model: nn.Module,
    loader: DataLoader,
    optimizer,
    scheduler,
    cfg: TrainConfig,
    train: bool,
) -> dict:
    model.train() if train else model.eval()
    device = cfg.device

    total_loss = 0.0
    all_ppf20_pred,  all_ppf20_tgt  = [], []
    all_rftn20_pred, all_rftn20_tgt = [], []

    ctx = torch.enable_grad() if train else torch.no_grad()
    with ctx:
        for batch in loader:
            ts     = batch["ts"].to(device)
            mask   = batch["padding_mask"].to(device)
            static = batch["static"].to(device)
            ppf20  = batch["ppf20"].to(device)
            rftn20 = batch["rftn20"].to(device)

            ppf20_pred, rftn20_pred = model(ts, static, mask)

            loss = (
                cfg.ppf20_weight  * weighted_mse(ppf20_pred,  ppf20)
              + cfg.rftn20_weight * weighted_mse(rftn20_pred, rftn20)
            )

            if train:
                optimizer.zero_grad()
                loss.backward()
                nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
                optimizer.step()

            total_loss += loss.item() * len(ppf20)
```

```
            all_ppf20_pred.append(ppf20_pred.detach().cpu().numpy())
            all_ppf20_tgt.append(ppf20.detach().cpu().numpy())
            all_rftn20_pred.append(rftn20_pred.detach().cpu().numpy())
            all_rftn20_tgt.append(rftn20.detach().cpu().numpy())

    n = len(loader.dataset)
    metrics = {"loss": total_loss / n}
    metrics.update(compute_metrics(
        np.concatenate(all_ppf20_pred),
        np.concatenate(all_ppf20_tgt),
        prefix="ppf20",
    ))
    metrics.update(compute_metrics(
        np.concatenate(all_rftn20_pred),
        np.concatenate(all_rftn20_tgt),
        prefix="rftn20",
    ))
    return metrics

def train(
    cfg: TrainConfig,
    train_dataset: Dataset,
    val_store: GPUValStore,
    ts_input_dim: int,
    static_input_dim: int,
):
    os.makedirs(cfg.checkpoint_dir, exist_ok=True)
    ckpt_path = os.path.join(cfg.checkpoint_dir, f"{cfg.run_name}_best.pt")

    checkpoint_count = 0

    model = AnesthesiaDrugPredictor(
        ts_input_dim=ts_input_dim,
        static_input_dim=static_input_dim,
        hidden_size=cfg.hidden_size,
        num_layers=cfg.num_layers,
        dropout=cfg.dropout,
```

```
        arch=cfg.arch,
        bidirectional=cfg.bidirectional,
    ).to(cfg.device)

    total_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
    print(f"\nModel parameters: {total_params:,}")
    print(model)

    optimizer = AdamW(model.parameters(), lr=cfg.lr,
weight_decay=cfg.weight_decay)
    scheduler = CosineAnnealingWarmRestarts(
        optimizer,
        T_0     = cfg.chunks_per_epoch,
        T_mult  = 1,
        eta_min = cfg.lr * 0.01
    )

    best_val_loss = float("inf")
    patience_ctr  = 0
    history       = []
    global_step   = 0
    last_val_m    = None

    print(f"\n{'Epoch':>6}  {'Chunk':>7}  {'TrnLoss':>9}  {'ValLoss':>9}  "
      f"{'PPF_MAE':>9}  {'RFT_MAE':>9}  {'PPF_R²':>7}  {'RFT_R²':>7}  "
      f"{'PPF_SpkMAE':>11}  {'RFT_SpkMAE':>11}  "
      f"{'PPF_Rec':>8}  {'RFT_Rec':>8}  {'Time':>7}")
    print("─" * 90)

    for epoch in range(1, cfg.max_epochs + 1):
        chunk_loaders = make_chunk_loaders(train_dataset, cfg)

        for chunk_idx, chunk_loader in enumerate(chunk_loaders, start=1):

          t0 = time.time()
```

```
        train_m = run_chunk(model, chunk_loader, optimizer, scheduler, cfg,
train=True)

        scheduler.step(global_step)

        global_step += 1

        is_val_chunk = (global_step % cfg.val_every_n_chunks == 0) or (chunk_idx
== len(chunk_loaders))

        if is_val_chunk:
           val_m   = val_store.evaluate(model, cfg)
           last_val_m = val_m
        else:
           val_m = last_val_m

        elapsed = time.time() - t0

        row = {"epoch": epoch, "train": train_m, "val": val_m, "chunk": chunk_idx,
"global_step" : global_step}
        history.append(row)

        chunk_label = f"{chunk_idx}/{len(chunk_loaders)}"
        if val_m is not None:
          def _fmt_spike(val_m, key):
             v = val_m.get(key, float("nan"))
             return f"{'-':>11}" if v != v else f"{v:>11.4f}"

          print(
             f"{epoch:>6}  {chunk_label:>7}  {train_m['loss']:>9.4f}
{val_m['loss']:>9.4f}  "
             f"{val_m['ppf20_mae']:>9.4f}  {val_m['rftn20_mae']:>9.4f}  "
             f"{val_m['ppf20_r2']:>7.4f}  {val_m['rftn20_r2']:>7.4f}  "
             f"{_fmt_spike(val_m, 'ppf20_spike_mae')}  {_fmt_spike(val_m,
'rftn20_spike_mae')}  "
             f"{val_m.get('ppf20_spike_recall', float('nan')):>8.3f}  "
             f"{val_m.get('rftn20_spike_recall', float('nan')):>8.3f}  "
```

```
            f"{elapsed:>6.1f}s"
          )
        else:
          print(
              f"{epoch:>6}  {chunk_label:>7}  {train_m['loss']:>8.4f}  "
              f"{'-':>8}  {'-':>8}  {'-':>8}  {'-':>7}  {'-':>7}  "
              f"{elapsed:>6.1f}s"
            )

        if is_val_chunk:
           if val_m["loss"] < best_val_loss:
              best_val_loss = val_m["loss"]
              patience_ctr  = 0
              torch.save({
                 "epoch": epoch,
                 "chunk": chunk_idx,
                 "global_step": global_step,
                 "model_state": model.state_dict(),
                 "optimizer_state": optimizer.state_dict(),
                 "val_loss": best_val_loss,
                 "config": asdict(cfg),
              }, os.path.join(cfg.checkpoint_dir,
f"{cfg.run_name}_{checkpoint_count}.pt"))
              print(f"        ✓ Saved checkpoint (val_loss={best_val_loss:.4f})")
              checkpoint_count += 1
           else:
              patience_ctr += 1
              if patience_ctr >= cfg.patience:
                 print(f"\nEarly stopping: no improvement for {patience_ctr} chunks "
                       f"(epoch {epoch}, chunk {chunk_idx})."
                 )
                 # Save history and return
                 history_path = os.path.join(cfg.checkpoint_dir,
f"{cfg.run_name}_history.json")
                 with open(history_path, "w") as f:
                    json.dump(history, f, indent=2)
```

```
                print(f"\nTraining history saved to {history_path}")
                print(f"Best checkpoint   : {ckpt_path}
(val_loss={best_val_loss:.4f})")
                shutil.copy(os.path.join(cfg.checkpoint_dir,
f"{cfg.run_name}_{checkpoint_count-1}.pt")
                        , os.path.join(cfg.checkpoint_dir, f"{cfg.run_name}_best.pt"))
                return model, history

    history_path = os.path.join(cfg.checkpoint_dir, f"{cfg.run_name}_history.json")
    with open(history_path, "w") as f:
        json.dump(history, f, indent=2)
    print(f"\nTraining history saved to {history_path}")
    print(f"Best checkpoint   : {ckpt_path}  (val_loss={best_val_loss:.4f})")
    shutil.copy(os.path.join(cfg.checkpoint_dir,
f"{cfg.run_name}_{checkpoint_count-1}.pt")
            , os.path.join(cfg.checkpoint_dir, f"{cfg.run_name}_best.pt"))

    return model, history



def evaluate(
    model: nn.Module,
    test_loader: DataLoader,
    cfg: TrainConfig,
) -> dict:
    print("\n── Test Set Evaluation
──────────────────────────────────────────────────")
    metrics = run_chunk(model, test_loader, None, None, cfg, train=False)
    for k, v in metrics.items():
        print(f"  {k:20s}: {v:.4f}")
    return metrics


def load_best_model(
    ckpt_path: str,
    ts_input_dim: int,
```

```
    static_input_dim: int,
    device: str = "cpu",
) -> AnesthesiaDrugPredictor:
    ckpt = torch.load(ckpt_path, map_location=device)
    cfg_dict = ckpt["config"]
    model = AnesthesiaDrugPredictor(
        ts_input_dim=ts_input_dim,
        static_input_dim=static_input_dim,
        hidden_size=cfg_dict["hidden_size"],
        num_layers=cfg_dict["num_layers"],
        dropout=cfg_dict["dropout"],
        arch=cfg_dict["arch"],
        bidirectional=cfg_dict["bidirectional"],
    ).to(device)
    model.load_state_dict(ckpt["model_state"])
    print(f"Loaded model from epoch {ckpt['epoch']}
(val_loss={ckpt['val_loss']:.4f})")
    return model

import sys, io

class TeeLogger:
    def __init__(self):
        self.buffer  = io.StringIO()
        self._stdout = sys.stdout

    def __enter__(self):
        sys.stdout = self
        return self

    def __exit__(self, *args):
        sys.stdout = self._stdout  # always restores, even on exception

    def write(self, msg):
        self._stdout.write(msg)
        self.buffer.write(msg)
```

```
    def flush(self):
        self._stdout.flush()

    def get(self):
        return self.buffer.getvalue()

data_in_memory = False
train_ds, val_store, test_loader, ts_features, static_features = None, None, None,
None, None



import os
import torch
import datetime

MODEL_NAME = "Informer_regressor"
RUN_NAME = "run_4"
RUN_DIR =
f"/content/drive/MyDrive/Mag2026Data/{MODEL_NAME}/{RUN_NAME}_differ
ent_loss_10"

CONFIG = dict(
    # Data
    data_dir    = "/content/drive/MyDrive/Mag2026Data/data",
    batch_size  = 64,
    window_size = 300,    # past timesteps: 60 × 5 s = 5 min
    horizon     = 4,     # future timesteps for mean target (= 30 s)
    target_type = "next", # mean, max, closeness_bias, next
    num_workers = 2,
    chunks_per_epoch = 10,
    val_every_n_chunks = 5,

    # Model
    arch            = "informer",
    hidden_size     = 256,
    num_layers      = 4,
```

```
    dropout        = 0.3,
    use_attention   = True,
    bidirectional   = True,

    lr           = 3e-4,
    weight_decay  = 1e-5,
    max_epochs    = 50,
    patience      = 7,
    ppf20_weight  = 0.5,
    rftn20_weight = 0.5,

    run_dir = RUN_DIR,
    checkpoint_dir = f"{RUN_DIR}/checkpoints/",
    run_name       = RUN_NAME,
)

CONFIG["batch_size"]  = 512
CONFIG["max_epochs"]  = 15
CONFIG["patience"]    = 2
CONFIG["num_workers"] = 4

def main(device, train_ds, val_store, test_loader, ts_features, static_features, config:
dict = CONFIG):
    cfg = TrainConfig(
        data_dir        = config["data_dir"],
        batch_size      = config["batch_size"],
        window_size     = config["window_size"],
        horizon         = config["horizon"],
        num_workers     = config["num_workers"],
        arch            = config["arch"],
        hidden_size     = config["hidden_size"],
        num_layers      = config["num_layers"],
        dropout         = config["dropout"],
        use_attention   = config["use_attention"],
        bidirectional   = config["bidirectional"],
        lr              = config["lr"],
        weight_decay    = config["weight_decay"],
```

```
        max_epochs      = config["max_epochs"],
        patience        = config["patience"],
        ppf20_weight    = config["ppf20_weight"],
        rftn20_weight   = config["rftn20_weight"],
        chunks_per_epoch = config["chunks_per_epoch"],
        val_every_n_chunks = config["val_every_n_chunks"],
        checkpoint_dir  = config["checkpoint_dir"],
        run_name        = config["run_name"],
        device          = device,
    )

    model, history = train(cfg, train_ds, val_store, ts_input_dim, static_input_dim)

    ckpt_path = os.path.join(cfg.checkpoint_dir, f"{cfg.run_name}_best.pt")
    best_model = load_best_model(ckpt_path, ts_input_dim, static_input_dim, device)
    evaluate(best_model, test_loader, cfg)

    return best_model, history



device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Device: {device}")
if not data_in_memory:
    train_ds, val_store, test_loader, ts_features, static_features = \
        build_dataloaders(
            data_dir    = CONFIG["data_dir"],
            batch_size  = CONFIG["batch_size"],
            window_size = CONFIG["window_size"],
            horizon     = CONFIG["horizon"],
            num_workers = CONFIG["num_workers"],
            target_type = CONFIG["target_type"],
            device      = device
        )
    data_in_memory = True

ts_input_dim     = len(ts_features)
```

```
static_input_dim = len(static_features)

if __name__ == "__main__":
    with TeeLogger() as logger:
        main(device, train_ds, val_store, test_loader, ts_features, static_features,
CONFIG)

        timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
        log_path  = f"{RUN_DIR}/logs/train_{timestamp}.txt"
        os.makedirs(os.path.dirname(log_path), exist_ok=True)

        with open(log_path, "w") as f:
            f.write(logger.get())

        print(f"Log saved → {log_path}")

config = CONFIG

cfg = TrainConfig(
        data_dir        = config["data_dir"],
        batch_size      = config["batch_size"],
        window_size     = config["window_size"],
        horizon         = config["horizon"],
        num_workers     = config["num_workers"],
        arch            = config["arch"],
        hidden_size     = config["hidden_size"],
        num_layers      = config["num_layers"],
        dropout         = config["dropout"],
        use_attention   = config["use_attention"],
        bidirectional   = config["bidirectional"],
        lr              = config["lr"],
        weight_decay    = config["weight_decay"],
        max_epochs      = config["max_epochs"],
        patience        = config["patience"],
        ppf20_weight    = config["ppf20_weight"],
        rftn20_weight   = config["rftn20_weight"],
        chunks_per_epoch = config["chunks_per_epoch"],
```

```
        val_every_n_chunks = config["val_every_n_chunks"],
        checkpoint_dir  = config["checkpoint_dir"],
        run_name        = config["run_name"],
        device          = device,
    )

ts_input_dim     = len(ts_features)
static_input_dim = len(static_features)
ckpt_path = os.path.join(cfg.checkpoint_dir, f"{cfg.run_name}_best.pt")
best_model = load_best_model(ckpt_path, ts_input_dim, static_input_dim, device)

import joblib
Rob = joblib.load(f"{CONFIG["data_dir"]}/rob_scaler.pkl")

test_loader.dataset[630]

test_loader.dataset[2510]

import matplotlib.pyplot as plt
import datetime

timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
image_path  = f"{RUN_DIR}/logs/"
os.makedirs(os.path.dirname(image_path), exist_ok=True)
ppf20_real = []
rftn20_real = []
ppf20_predict = []
rftn20_predict = []

best_model.eval()

with torch.no_grad():
    for i in range(620,2510):
        data_instance = test_loader.dataset[i]
        past_instance = test_loader.dataset[i]
```

```
        real_values =
Rob.inverse_transform([[float(past_instance["ppf20"]),float(past_instance["rftn20"]),
0,0]])

        # Real values
        ppf20_real.append(real_values[0][0])
        rftn20_real.append(real_values[0][1])

        # Prediction
        prediction = best_model.forward(
            data_instance['ts'].reshape(1, 300, 17).to(device),
            data_instance['static'].reshape(1, 45).to(device),
            data_instance['padding_mask'].reshape(1, 300).to(device)
        )

        predicted_values =
Rob.inverse_transform([[float(prediction[0]),float(prediction[1]),0,0]])



        ppf20_predict.append(predicted_values[0][0])
        rftn20_predict.append(predicted_values[0][1])

plt.figure(figsize=(14, 5))

plt.plot(ppf20_real, label="PPF20 Real")
plt.plot(ppf20_predict, label="PPF20 Predicted")

plt.title("PPF20: Real vs Predicted")
plt.xlabel("Sample")
plt.ylabel("PPF20")
plt.legend()
plt.grid(True)

plt.savefig(f"{image_path}ppf20_plot{timestamp}.png", dpi=300)
plt.show()

plt.figure(figsize=(14, 5))
```

```
plt.plot(rftn20_real, label="RFTN20 Real")
plt.plot(rftn20_predict, label="RFTN20 Predicted")

plt.title("RFTN20: Real vs Predicted")
plt.xlabel("Sample")
plt.ylabel("RFTN20")
plt.legend()
plt.grid(True)

plt.savefig(f"{image_path}rftn20_plot{timestamp}.png", dpi=300)
plt.show()

import matplotlib.pyplot as plt
import datetime

timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
image_path  = f"{RUN_DIR}/logs/"
os.makedirs(os.path.dirname(image_path), exist_ok=True)
ppf20_real = []
rftn20_real = []
ppf20_predict = []
rftn20_predict = []

best_model.eval()

with torch.no_grad():
    for i in range(0,10000):
        data_instance = test_loader.dataset[i]
        past_instance = test_loader.dataset[i]

        ppf20_real.append(float(past_instance["ppf20"]))
        rftn20_real.append(float(past_instance["rftn20"]))

        prediction = best_model.forward(
            data_instance['ts'].reshape(1, 300, 17).to(device),
            data_instance['static'].reshape(1, 45).to(device),
```

```
            data_instance['padding_mask'].reshape(1, 300).to(device)
        )


        ppf20_predict.append(float(prediction[0]))
        rftn20_predict.append(float(prediction[1]))

plt.figure(figsize=(14, 5))

plt.plot(ppf20_real, label="PPF20 Real")
plt.plot(ppf20_predict, label="PPF20 Predicted")

plt.title("PPF20: Real vs Predicted")
plt.xlabel("Sample")
plt.ylabel("PPF20")
plt.legend()
plt.grid(True)

plt.show()

plt.figure(figsize=(14, 5))

plt.plot(rftn20_real, label="RFTN20 Real")
plt.plot(rftn20_predict, label="RFTN20 Predicted")

plt.title("RFTN20: Real vs Predicted")
plt.xlabel("Sample")
plt.ylabel("RFTN20")
plt.legend()
plt.grid(True)

plt.show()
```