Министерство науки и высшего образования Российской Федерации

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)

Институт прикладной информатики и компьютерных наук

ОТЧЕТ

По дисциплине «Глубинное обучение»

Задание 2 «Разработка среды с неопределённостью»

Студент

Группы 932101

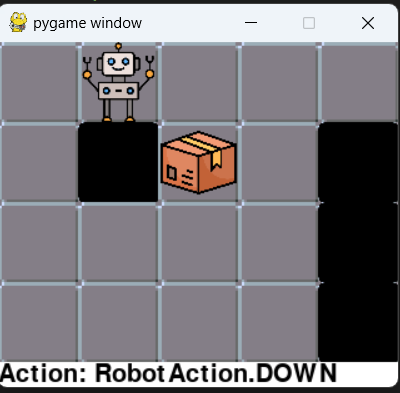
Насутион Рафли Аулиа Ризки

Томск – 2025

# **Цель работы**

Разработать среду на базе класса gym.env, позволяющую делать эксперименты с агентом – роботом, доставщиком посылки в районе города.

1. **Карту местности, для которой строится среда.**

****

**Карта местности**

На карте представлена сетка размером 5x4, которая содержит логистический центр (начальное положение робота), цель (посылка) и препятствия. Робот начинает в левом верхнем углу карты (ячейка [0, 0]). Цель и препятствия генерируются случайным образом при каждом запуске среды, при этом гарантируется, что:

* Цель и препятствия не занимают одно и то же место.
* Препятствия не блокируют путь робота к цели.

Пример карты:

* 🟦: Пустая ячейка
* 🤖: Робот
* 📦: Цель (посылка)
* 🟥: Препятствие

1. **Описание состояний и их визуализация**

Состояние определяется текущим положением робота в сетке. Например, если робот находится в ячейке [1, 2], состояние будет "(1, 2)". Для визуализации используется библиотека pygame, которая отображает карту с роботом, посылкой и препятствиями.

Пример визуализации:

🟦🟦🟦🟦🟦

🟦🤖🟦🟥🟦

🟦🟦🟦🟥📦

🟦🟦🟦🟦🟦

🟦🟦🟦🟦🟦

1. **Описание функции перехода**

Робот может двигаться в четырёх направлениях:

* Вверх
* Вниз
* Влево
* Вправо

Переходы между состояниями зависят от следующих условий:

1. Если новое положение находится вне границ карты, движение не происходит.
2. Если новое положение совпадает с позицией препятствия, движение не происходит.

Пример код реализации функции перехода:

    def perform\_action(self, robot\_action: RobotAction) -> bool:

        self.last\_action = robot\_action

        new\_pos = self.robot\_pos[:]

        # Определение нового положения робота в зависимости от действия

        if robot\_action == RobotAction.LEFT:

            new\_pos[1] -= 1

        elif robot\_action == RobotAction.RIGHT:

            new\_pos[1] += 1

        elif robot\_action == RobotAction.UP:

            new\_pos[0] -= 1

        elif robot\_action == RobotAction.DOWN:

            new\_pos[0] += 1

        # Проверка, является ли новое положение допустимым

        if self.is\_valid\_position(new\_pos):

            self.robot\_pos = new\_pos

        # Возвращает True, если робот достиг цели

        return self.robot\_pos == self.target\_pos

    # Метод для выполнения действия в среде

    def step(self, action):

        # Выполнение действия роботом и проверка достижения цели

        target\_reached = self.delivery\_robot.perform\_action(wr.RobotAction(action))

        # Проверка столкновения с препятствием

        robot\_pos = self.delivery\_robot.robot\_pos

        if tuple(robot\_pos) in self.current\_obstacles:

            reward = -1  # Штраф за столкновение

            terminated = True  # Завершаем эпизод

        else:

            reward = 0  # Нет награды или штрафа

            terminated = False

            if target\_reached:

                reward = 1  # Награда за достижение цели

                terminated = True  # Завершаем эпизод

        # Формируем наблюдение: объединяем позиции робота и цели

        obs = np.concatenate((self.delivery\_robot.robot\_pos, self.delivery\_robot.target\_pos))

        info = {}  # Дополнительная информация

        # Отображаем действие и визуализируем среду, если режим 'human'

        if self.render\_mode == 'human':

            print(wr.RobotAction(action))  # Печатаем действие

            self.render()

        return obs, reward, terminated, False, info  # Возвращаем результат

1. **Описание функции вознаграждения**

В задаче робот получает вознаграждение в зависимости от своей текущей позиции и выполненного действия:

* Если робот сталкивается с препятствием, он получает штраф -1, и эпизод заканчивается.
* Если робот достигает цели, он получает награду 1, и эпизод заканчивается.
* За каждое движение без достижения цели или столкновения робот не получает ни награды, ни штрафа (0).  
  Эта функция вознаграждения направлена на минимизацию столкновений и достижение цели за наименьшее количество шагов.

Пример код реализации функции вознаграждения:

def step(self, action):

    # Выполнение действия роботом

    target\_reached = self.delivery\_robot.perform\_action(wr.RobotAction(action))

    # Проверяем, столкнулся ли робот с препятствием

    robot\_pos = self.delivery\_robot.robot\_pos

    if tuple(robot\_pos) in self.current\_obstacles:

        reward = -1  # Штраф за столкновение с препятствием

        terminated = True  # Эпизод завершен

    else:

        reward = 0  # Нет награды или штрафа

        terminated = False

        if target\_reached:

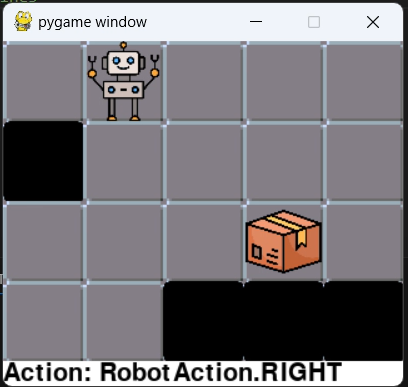
            reward = 1  # Награда за достижение цели

            terminated = True  # Эпизод завершен

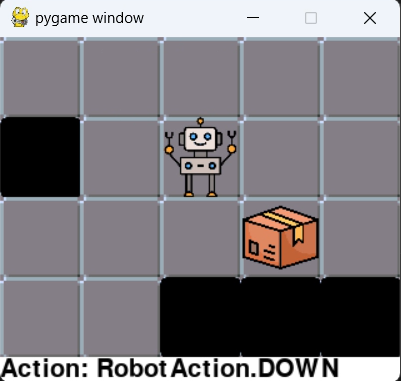
    # Возвращаем состояние, награду и индикатор завершения

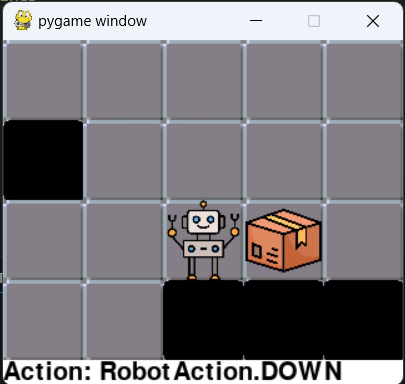
    return obs, reward, terminated, False, info

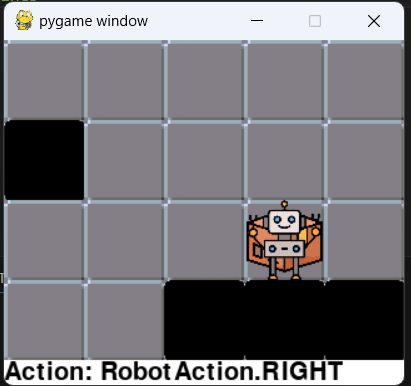
1. **Пример робота, работающего после обучения с использованием методов Q-Learning и StableBaseline3**

****

****

****

****

****

Wrapping the env with a `Monitor` wrapper

Wrapping the env in a DummyVecEnv.

R \_ \_ \_ \_

O \_ \_ \_ \_

\_ \_ \_ T \_

\_ \_ O O O

RobotAction.RIGHT

\_ R \_ \_ \_

O \_ \_ \_ \_

\_ \_ \_ T \_

\_ \_ O O O

RobotAction.RIGHT

\_ \_ R \_ \_

O \_ \_ \_ \_

\_ \_ \_ T \_

\_ \_ O O O

RobotAction.DOWN

\_ \_ \_ \_ \_

O \_ R \_ \_

\_ \_ \_ T \_

\_ \_ O O O

RobotAction.DOWN

\_ \_ \_ \_ \_

O \_ \_ \_ \_

\_ \_ R T \_

\_ \_ O O O

RobotAction.RIGHT

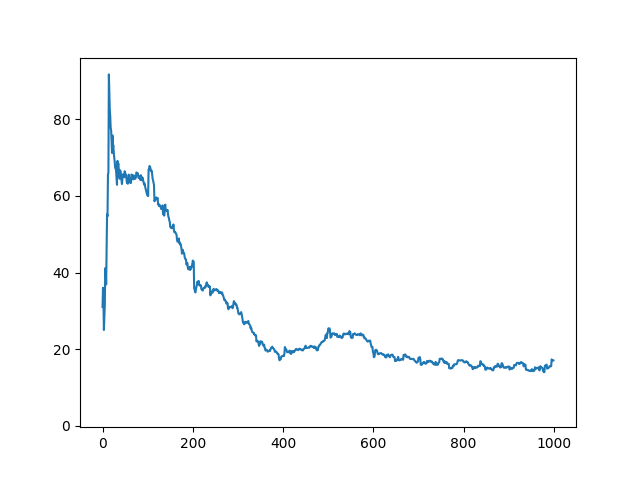
\_ \_ \_ \_ \_

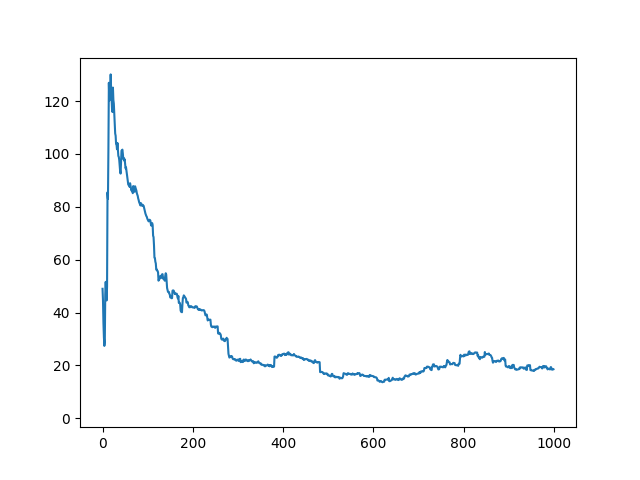
O \_ \_ \_ \_

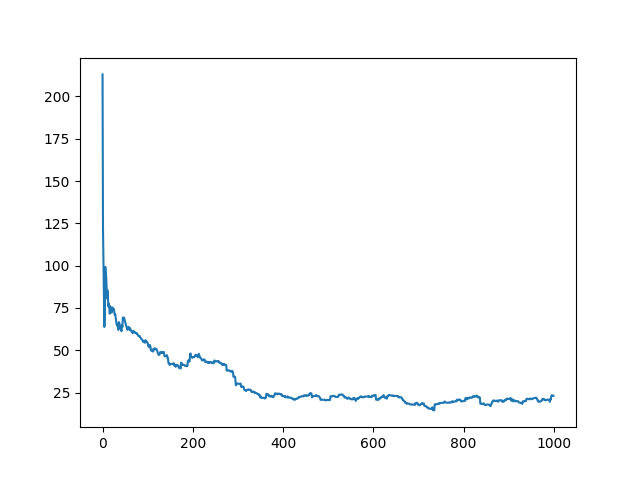
\_ \_ \_ R \_

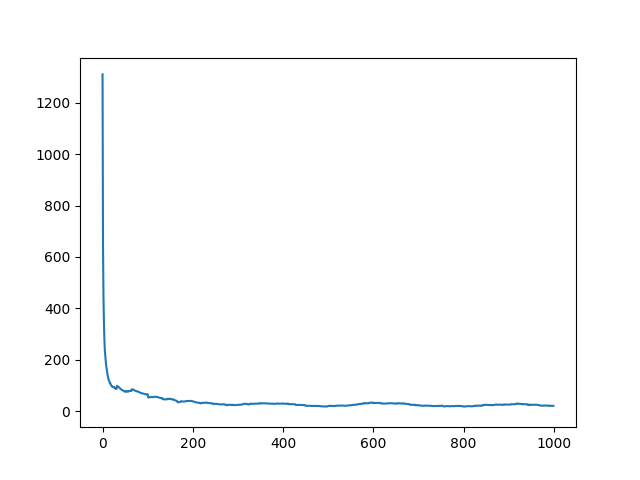
\_ \_ O O O

**DIAGRAM VISUALISASI YG SUDAH DI TRAIN DENGAN METODE Q-LEARNING**

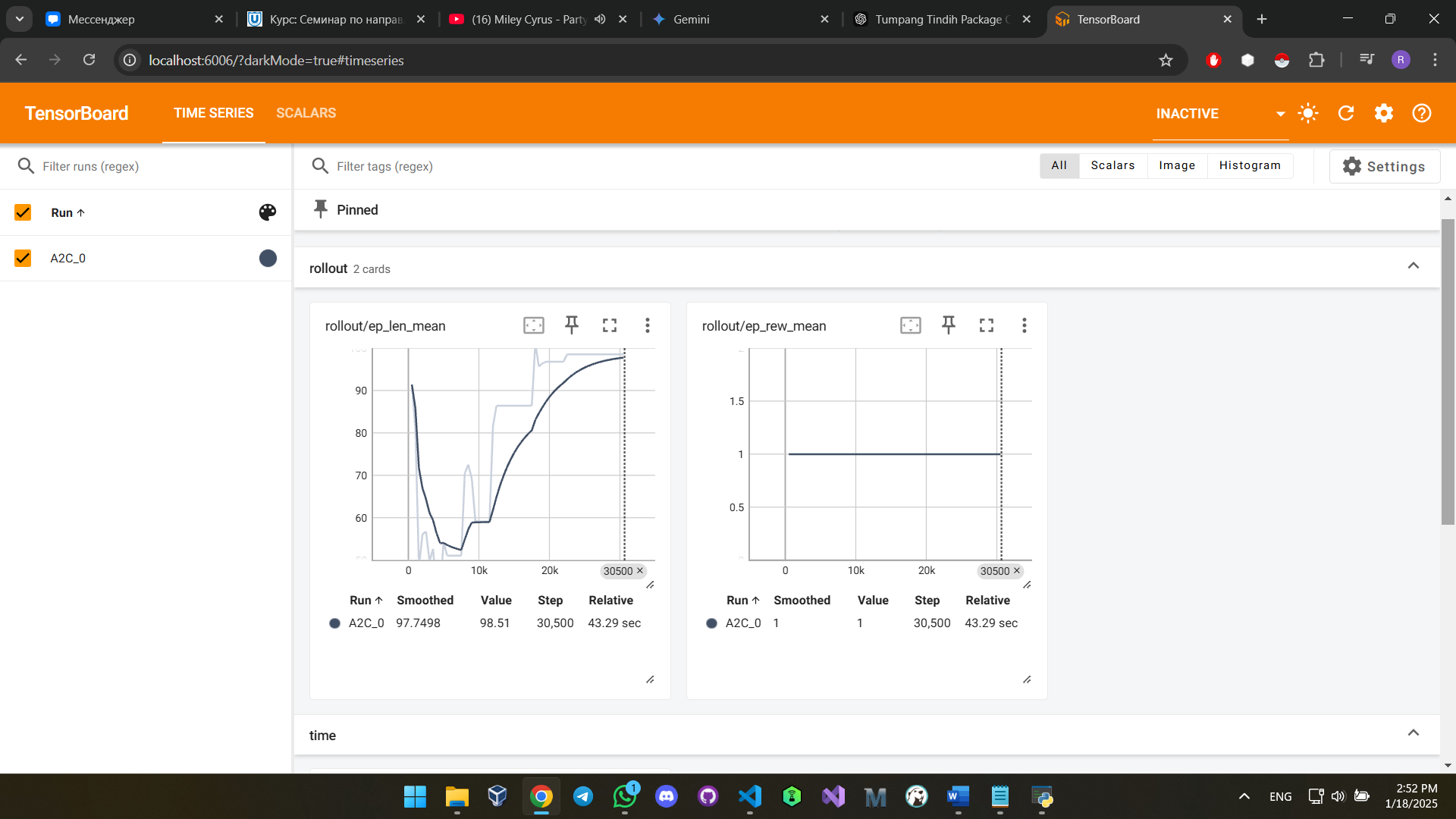


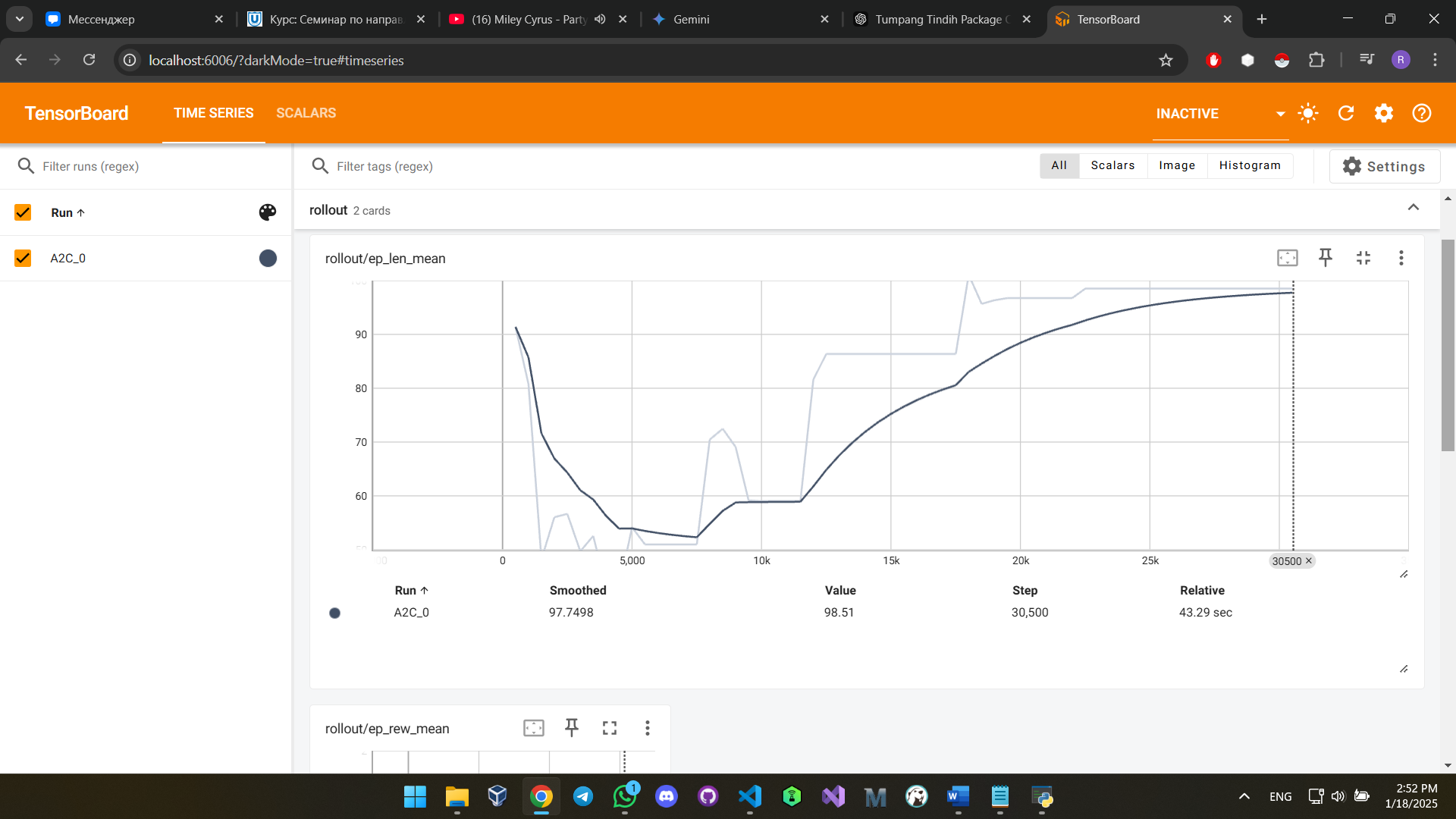


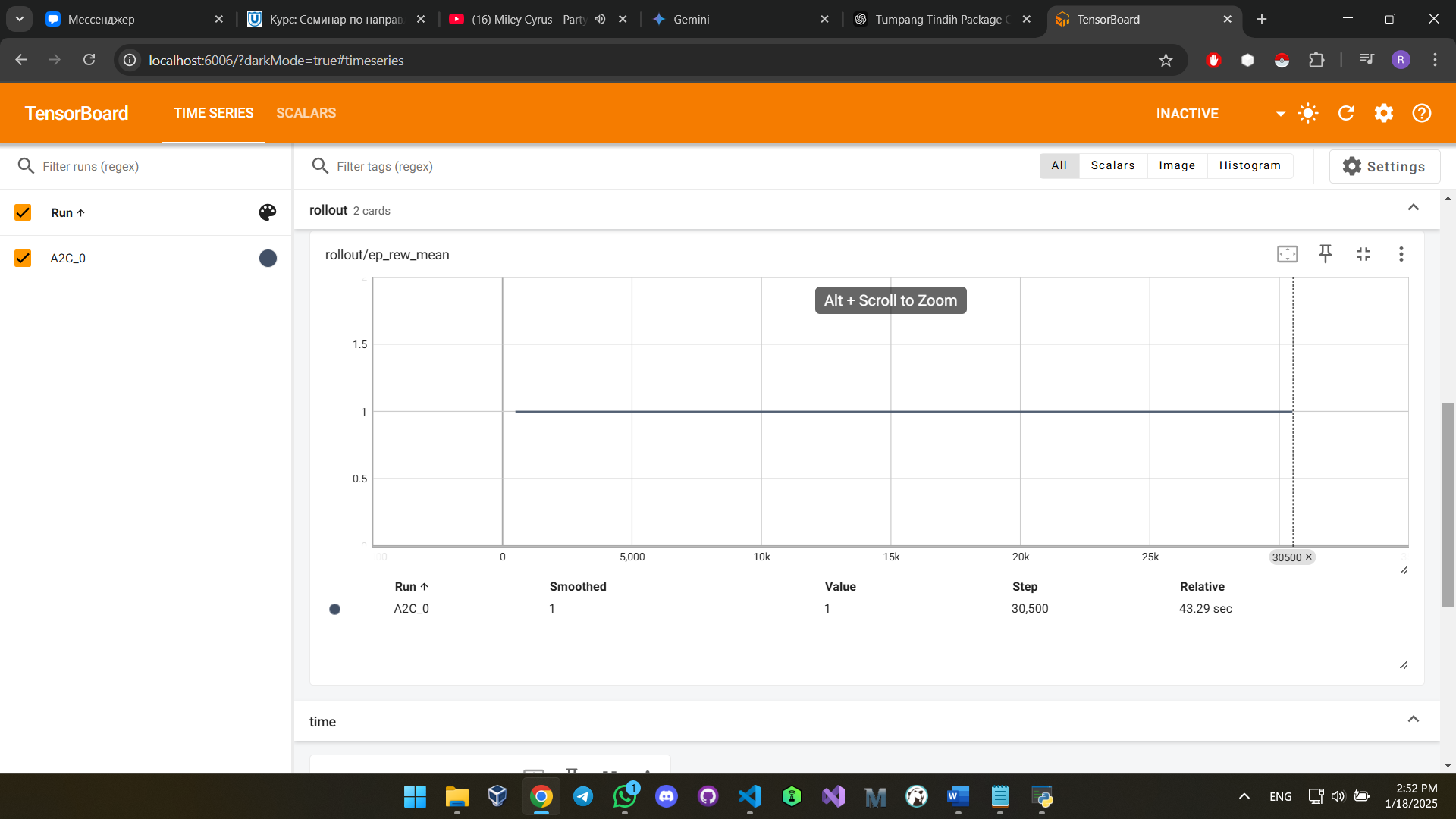


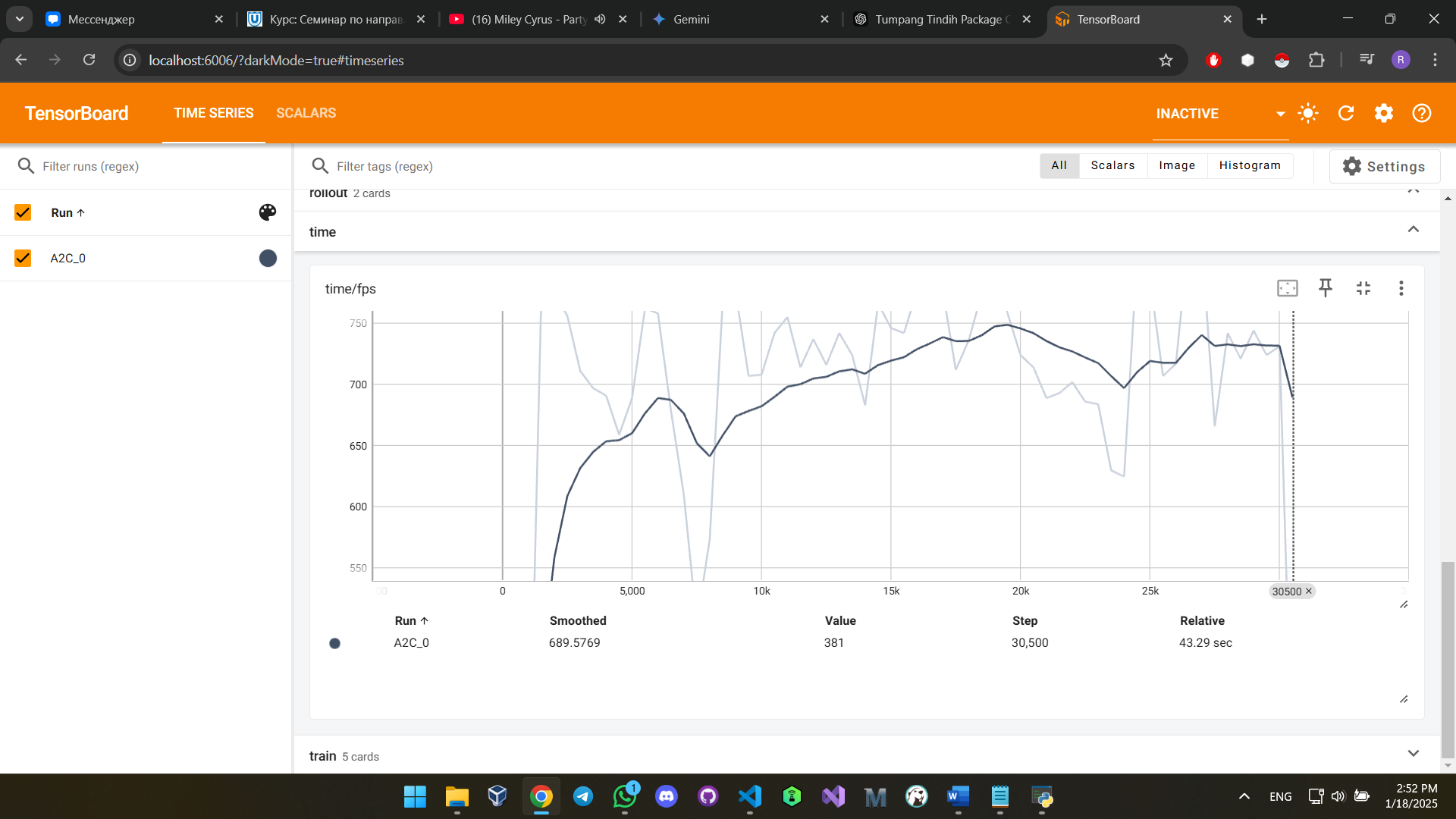


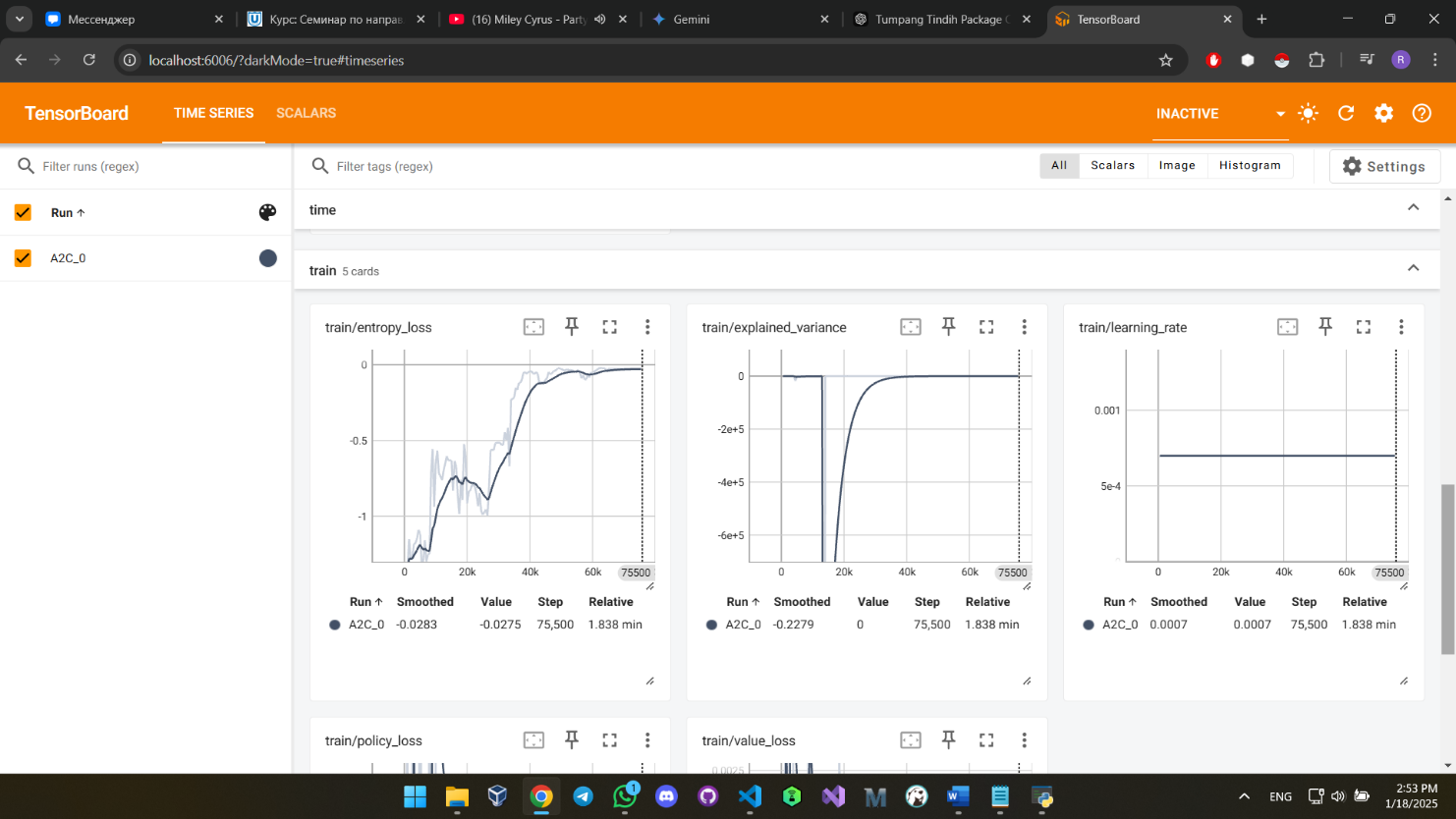
**HASIL TENSORBOARD YG SUDAH DI TRAIN DENGAN STABLEBASELINES3**

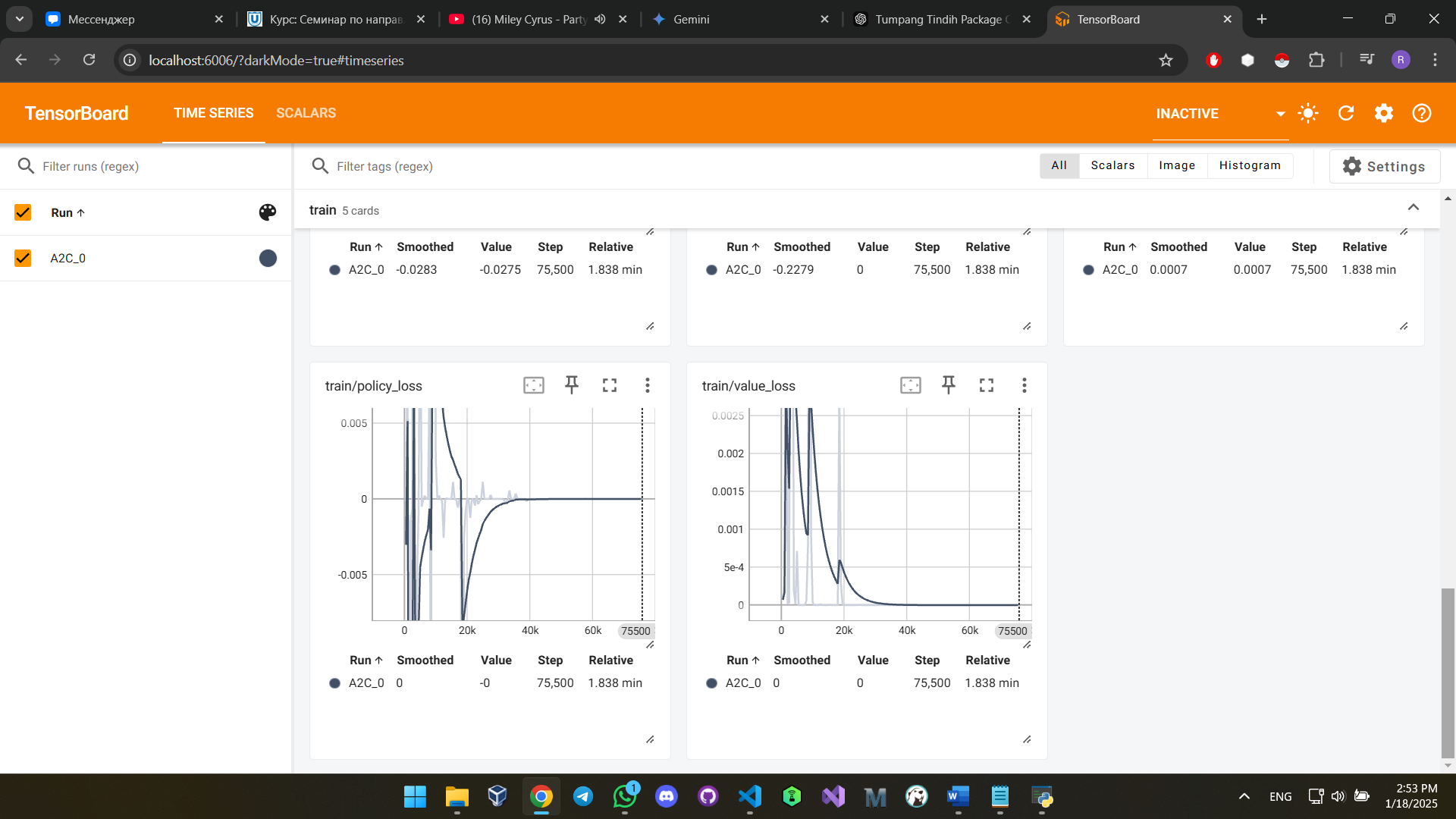


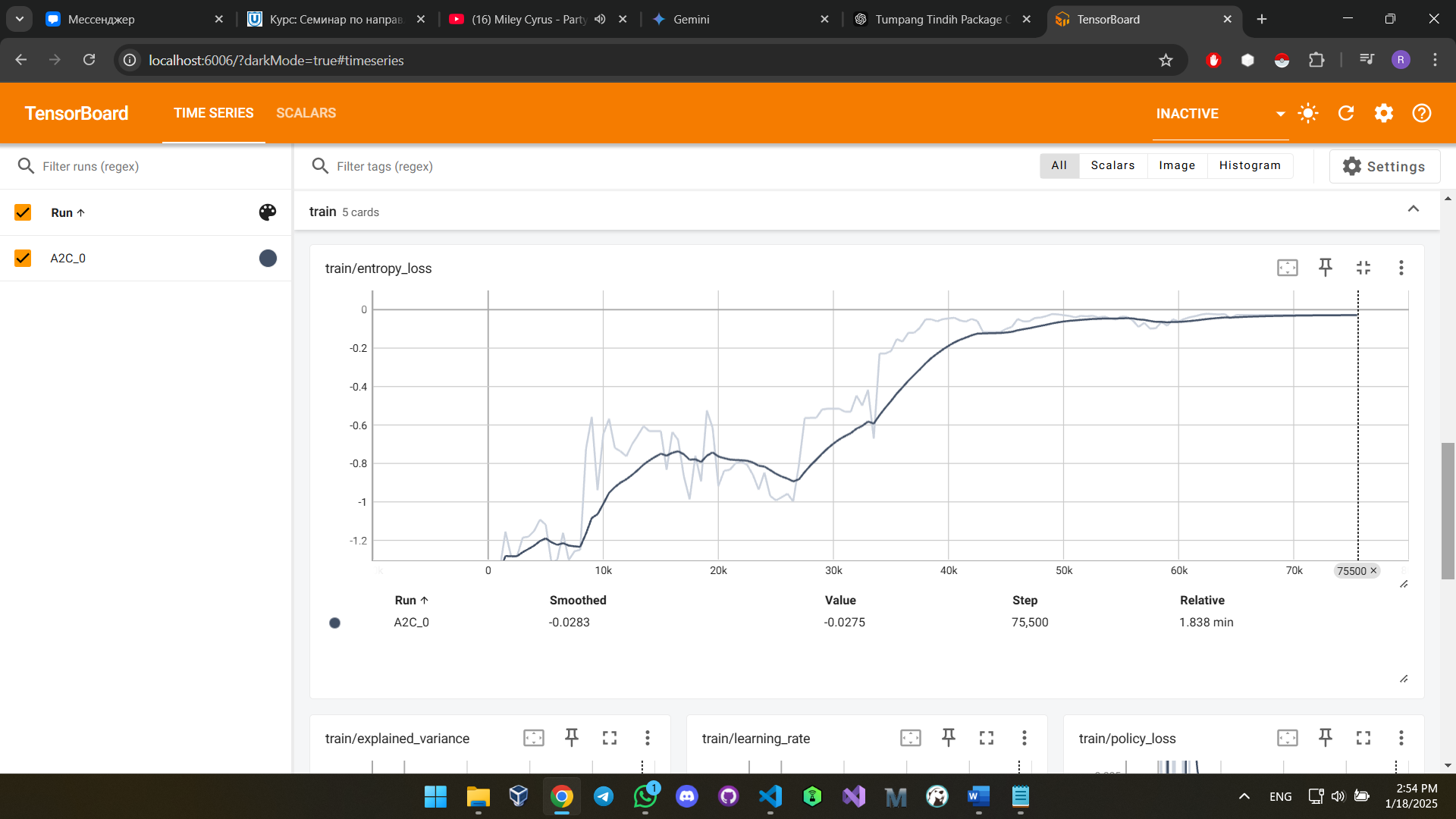


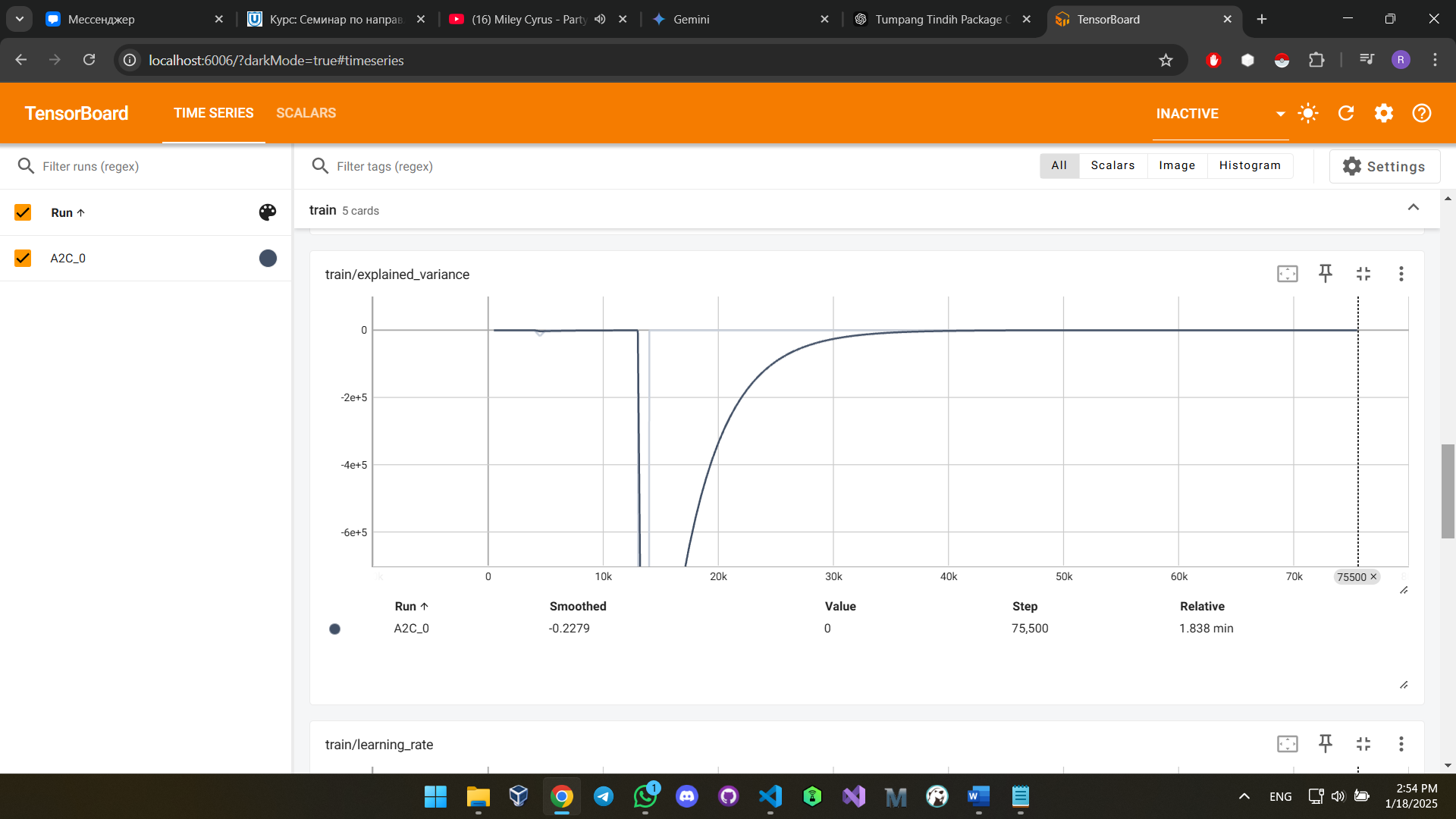


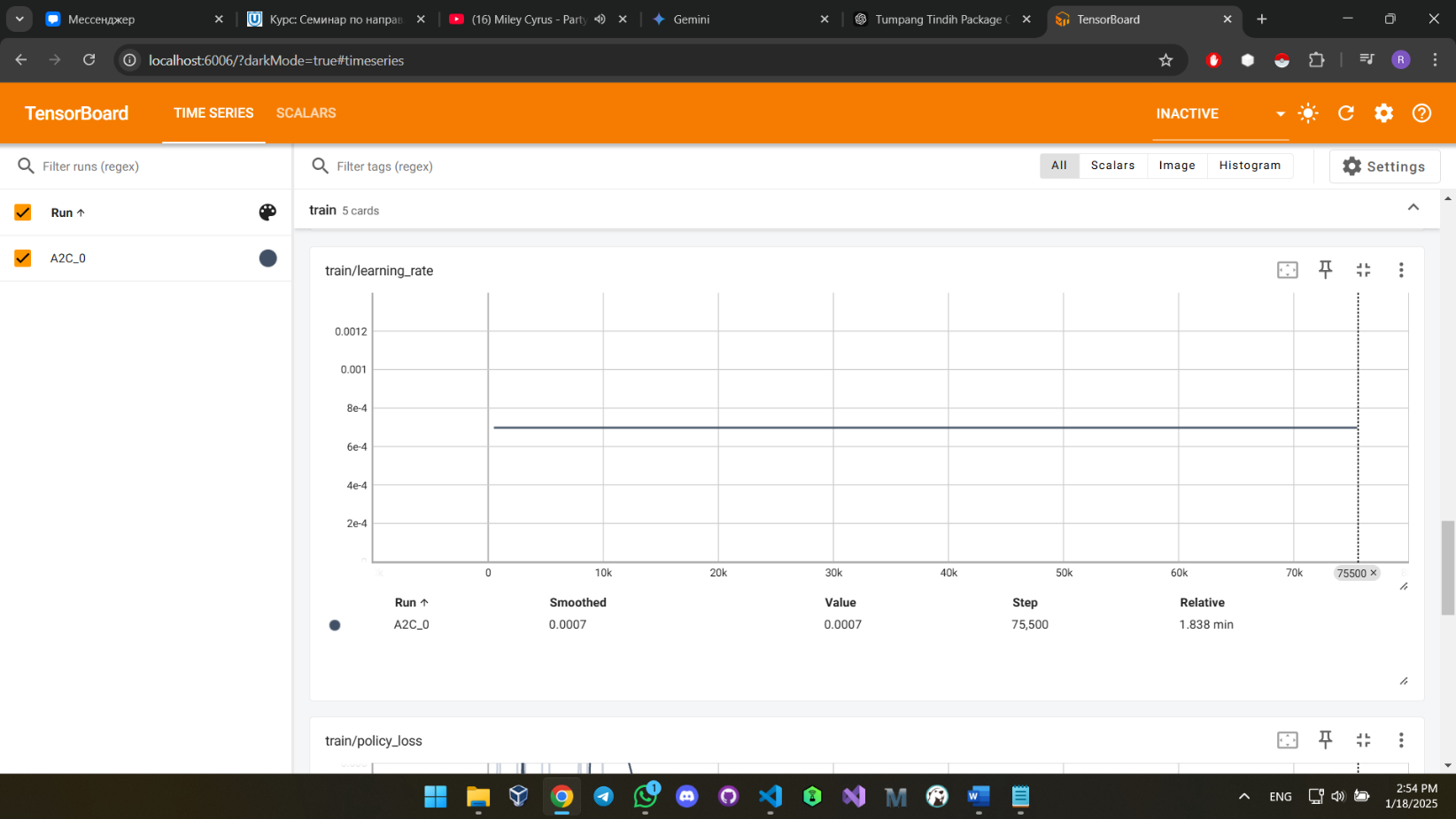


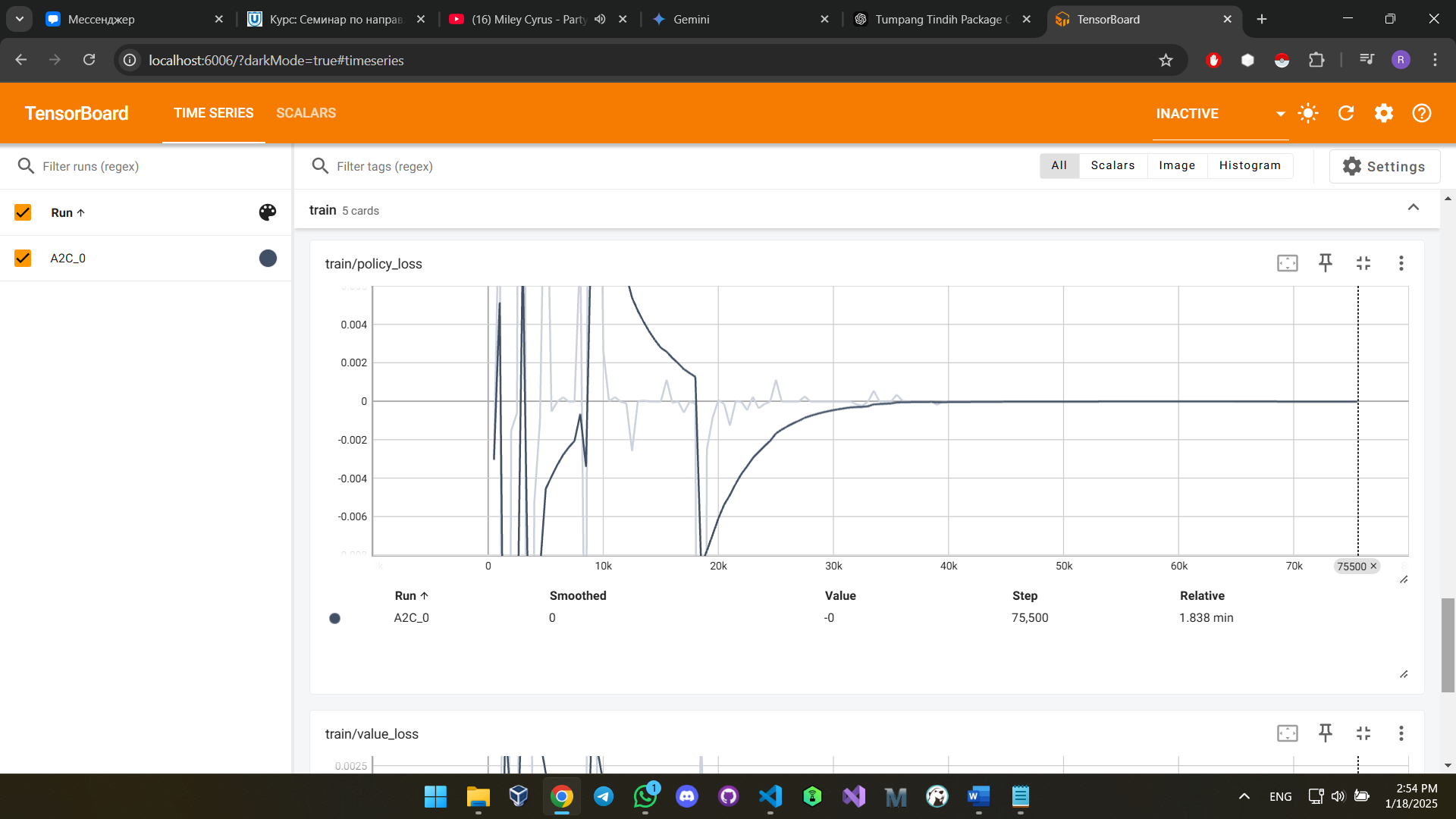


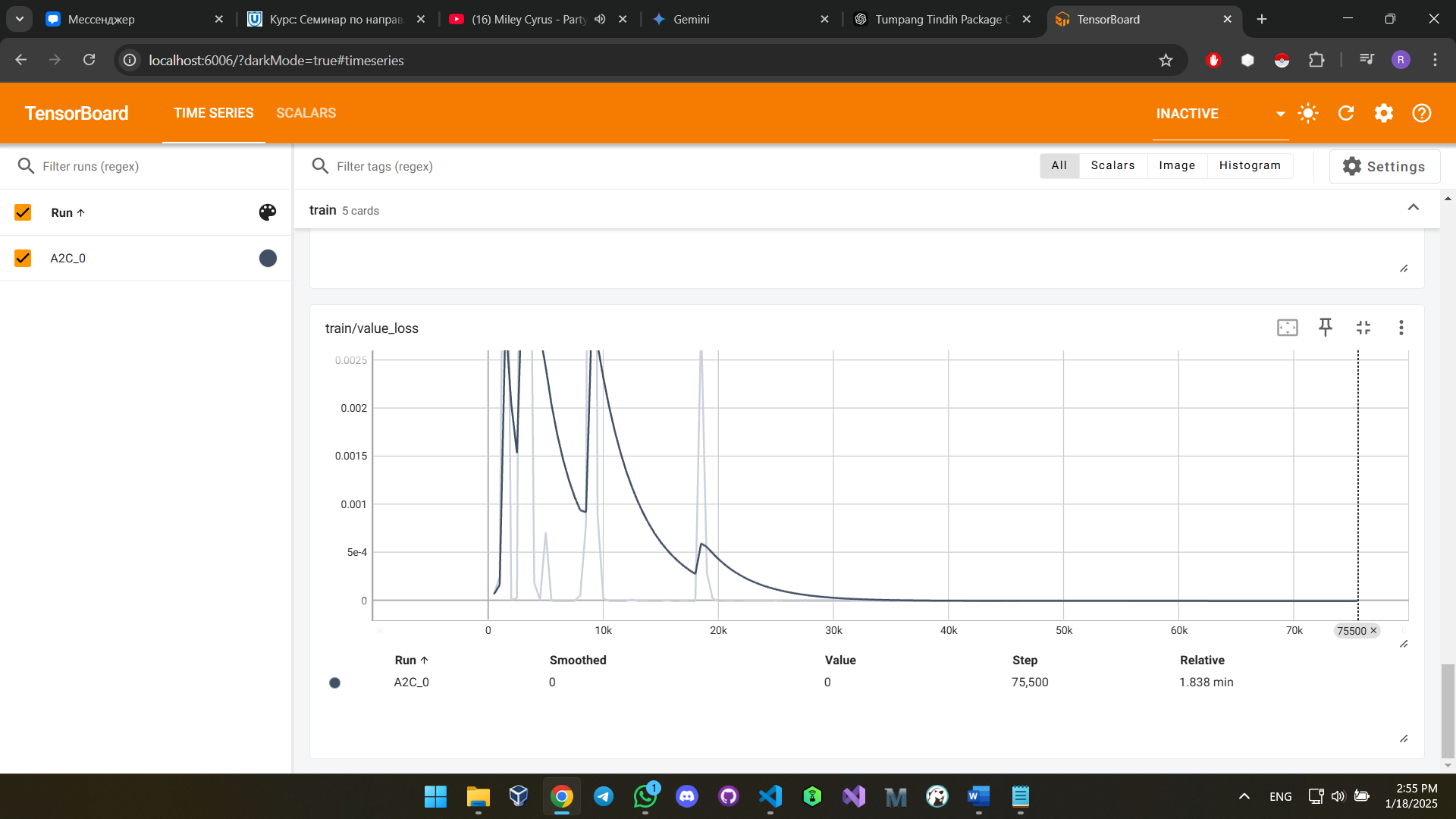












**PROGRAM CODE**