Project 1

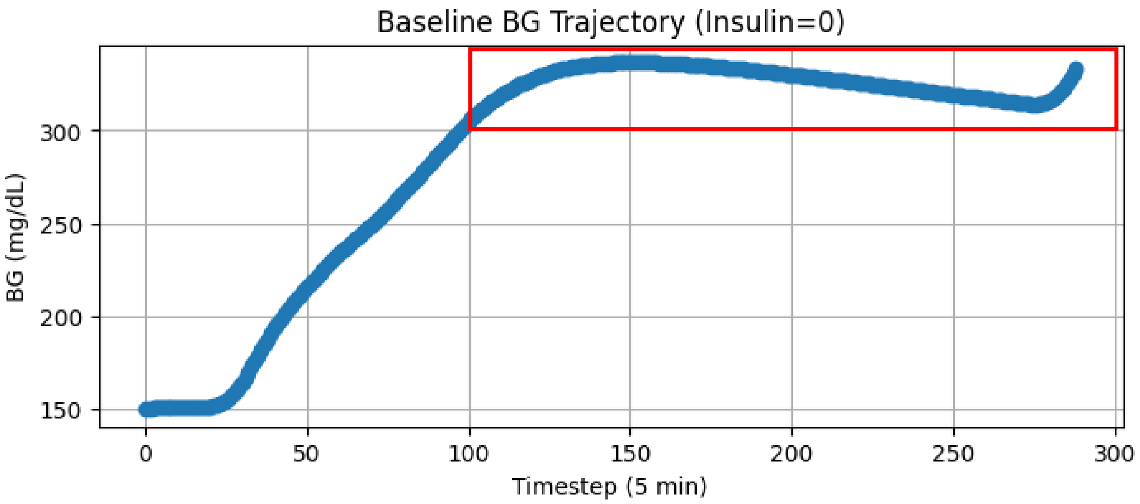
Blood Glucose (BG) Prediction and Control in Diabetes

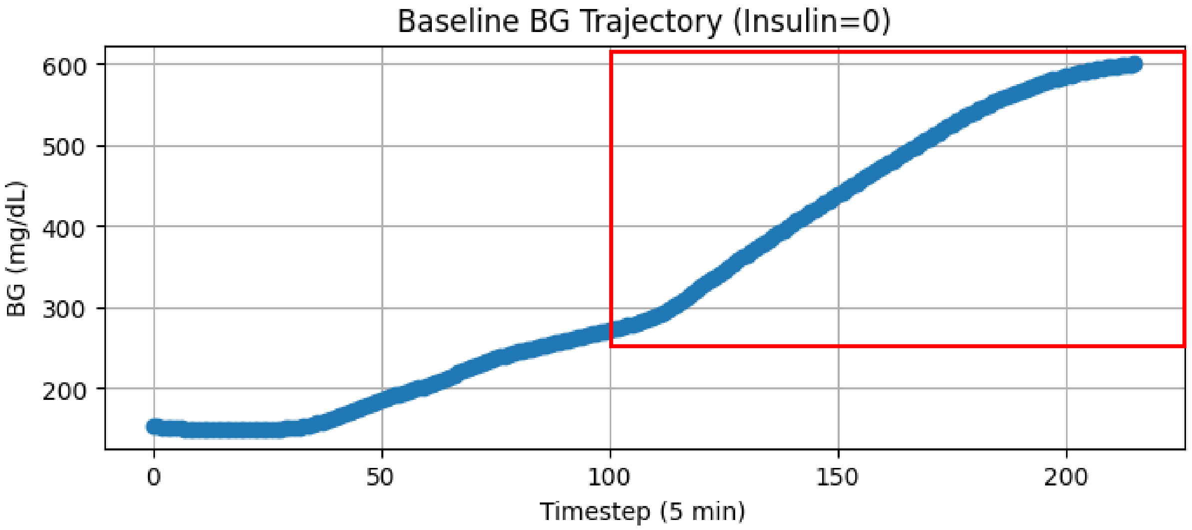
Yonsei University
김호연, 송성모, 최예찬

Contents:

1. Problem Statement.
- Prediction and Control of Blood Glucose in Type 1 Diabetes Patients

2. Simulation Environments.
- Glucose-Insulin system (Sim-glucose with OpenAI Gym framework)
- Activity and Energy intake patterns changing each time for the same patient.



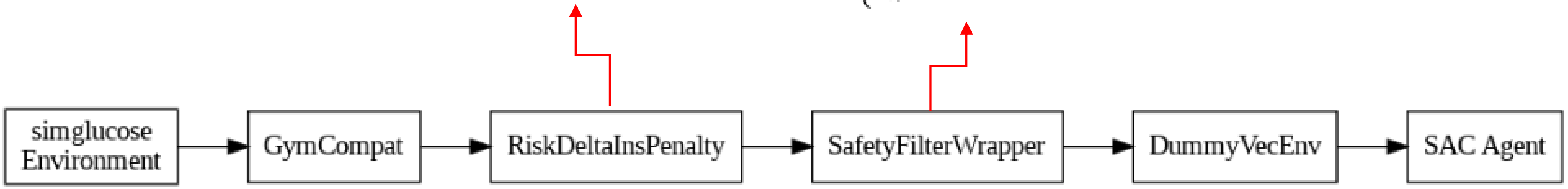


3. Summary of Models.

3.1. SAC with Filter, PPO with LGBM.

- Using SAC model with Risk Delta Function and Action Filter.
- Using PPO model with LightGBM Predictor.

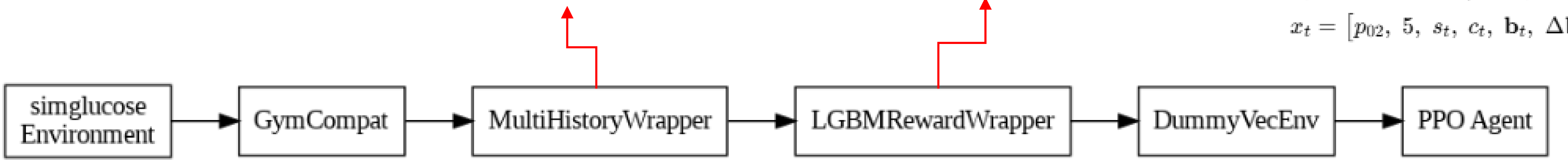
$$r = s\left(R_{t-1} - R_t\right) - \lambda_{\text{ins}} \times I_t \qquad a_t' = \begin{cases} s \times a_t, & \text{if } \mathrm{BG}_t < \text{cutoff}, \\ a_t, & \text{otherwise.} \end{cases}$$



(a) SAC model with Risk Delta Function and Action Filter

$$m_t = 60 \cdot t_{\text{hour}} + t_{\text{minute}}, \quad \alpha_t = \frac{2\pi\, m_t}{1440},$$
$$s_t = \sin(\alpha_t), \quad c_t = \cos(\alpha_t),$$
$$\mathbf{b}_t = [\mathrm{bg}_t,\ \mathrm{bg}_{t-1}, \ldots, \mathrm{bg}_{t-H}], \quad \Delta\mathbf{b}_t = \mathrm{diff}(\mathbf{b}_t),$$
$$\mathbf{i}_t = \text{hist\_insulin}, \quad \mathbf{m}_t = \text{hist\_meal},$$
$$x_t = [p_{02},\ 5,\ s_t,\ c_t,\ \mathbf{b}_t,\ \Delta\mathbf{b}_t,\ \mathbf{i}_t,\ \mathbf{m}_t].$$

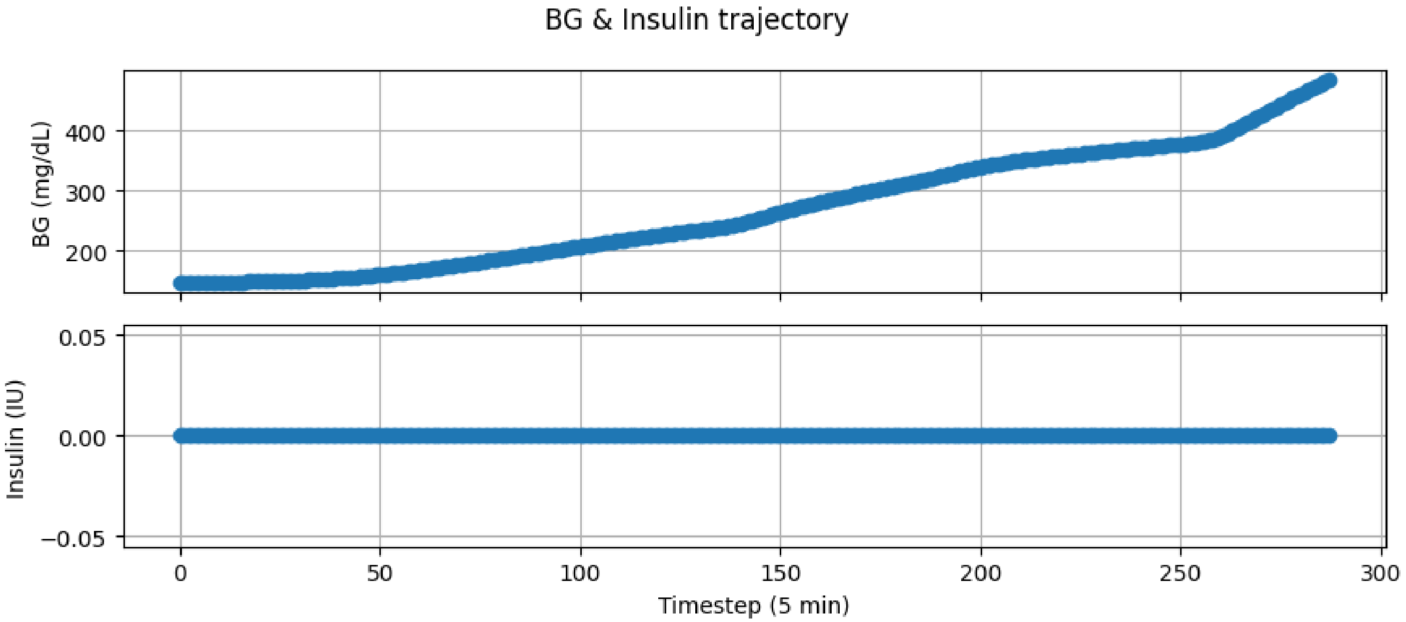
$$C_t = [o_{t-H+1},\ o_{t-H+2},\ \ldots,\ o_t] \qquad y_t = (\text{Regressor} \circ \text{Transformer})(x_t).$$



(b) PPO model with LightGBM Predictor

3. Summary of Models.
3.1. SAC with Filter, PPO with LGBM.

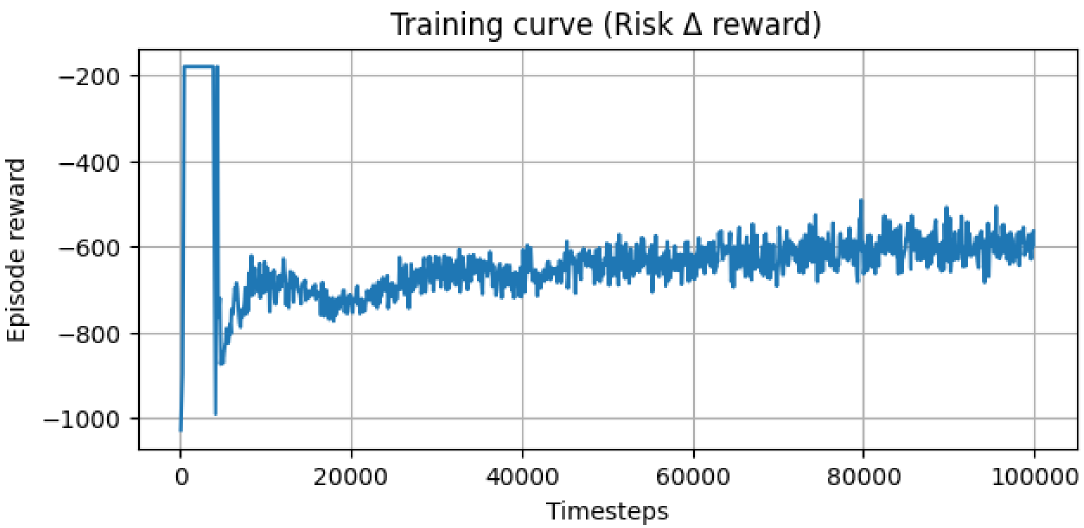
1) Navie SAC
→ No insulin administration.

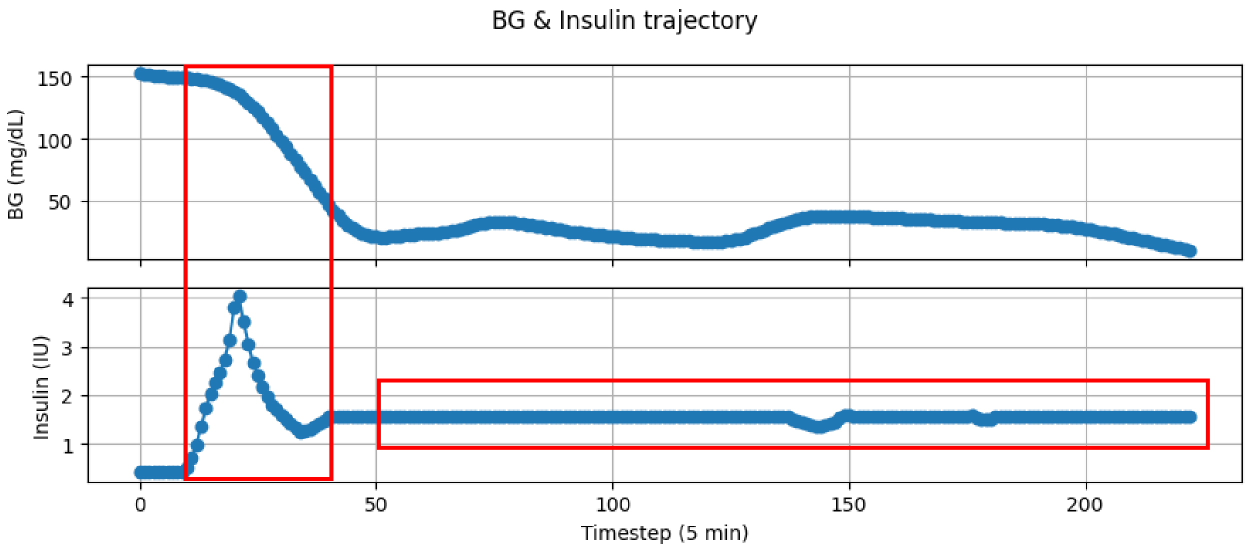


Blood Glucose (BG) Prediction and Control in Diabetes

3. Summary of Models.
3.1. SAC with Filter, PPO with LGBM.

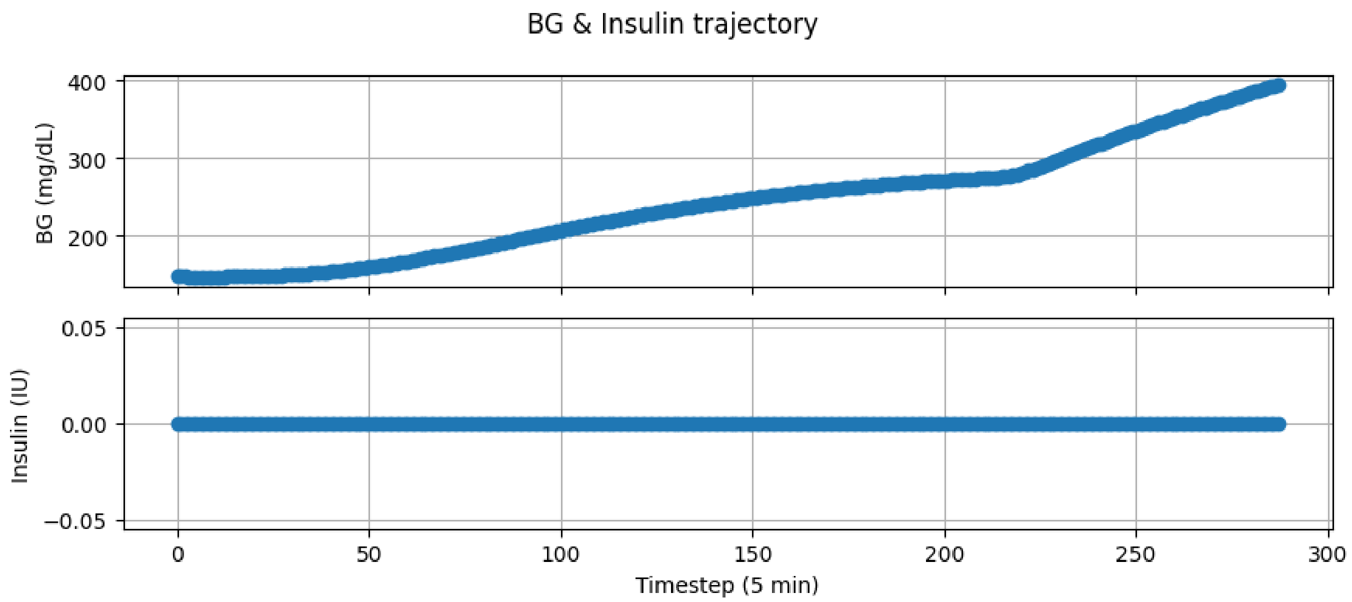
2) SAC with Filter
→ Adjusting insulin dosage, but using too much dose.
→ Risk Delta works well, but Action Filter does not work.





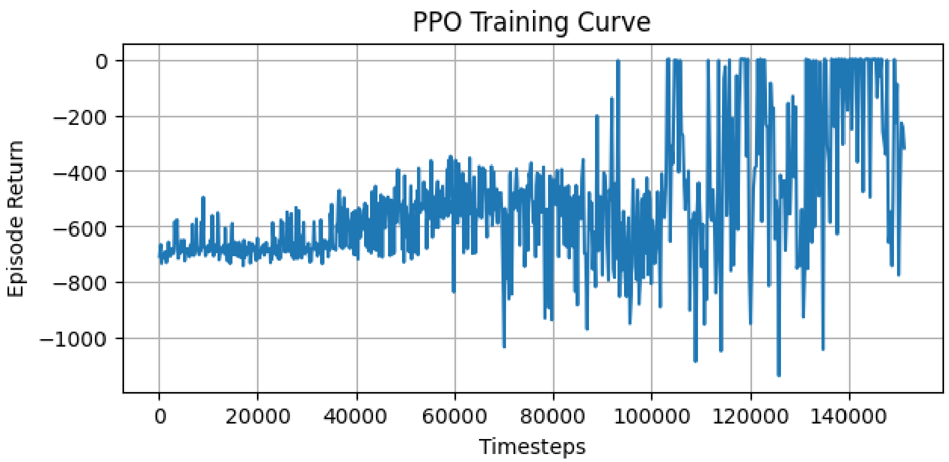
3. Summary of Models.
3.1. SAC with Filter, PPO with LGBM.

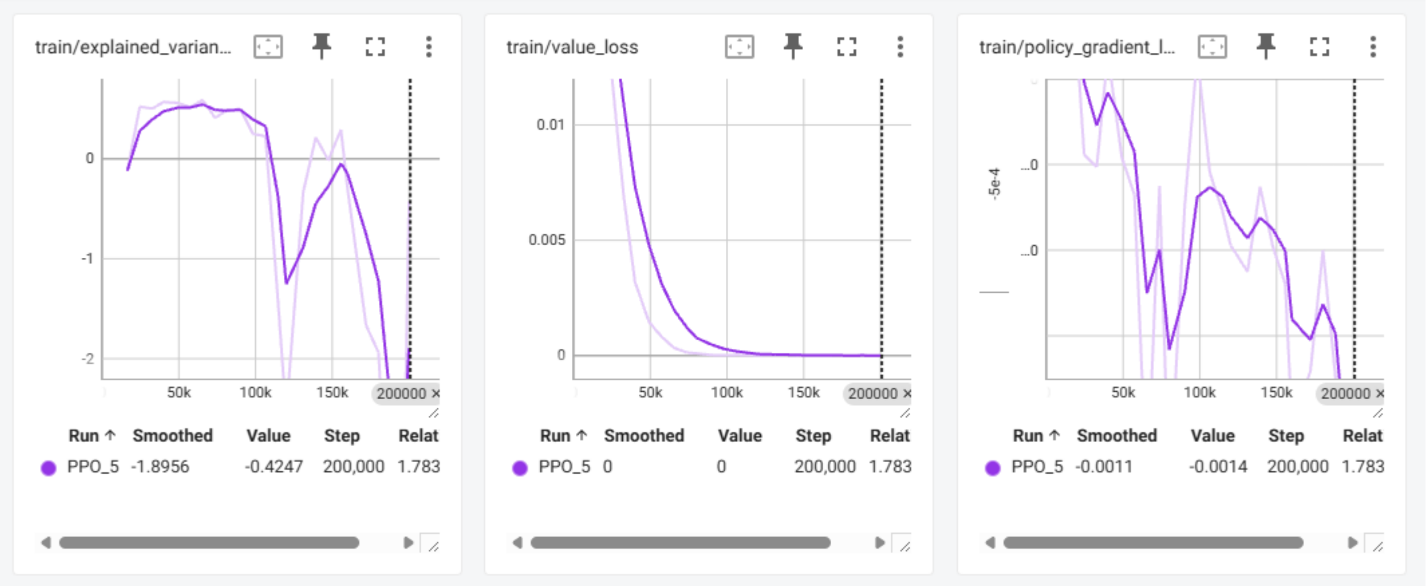
3) Navie PPO
→ No insulin administration.



3. Summary of Models.
3.1. SAC with Filter, PPO with LGBM.

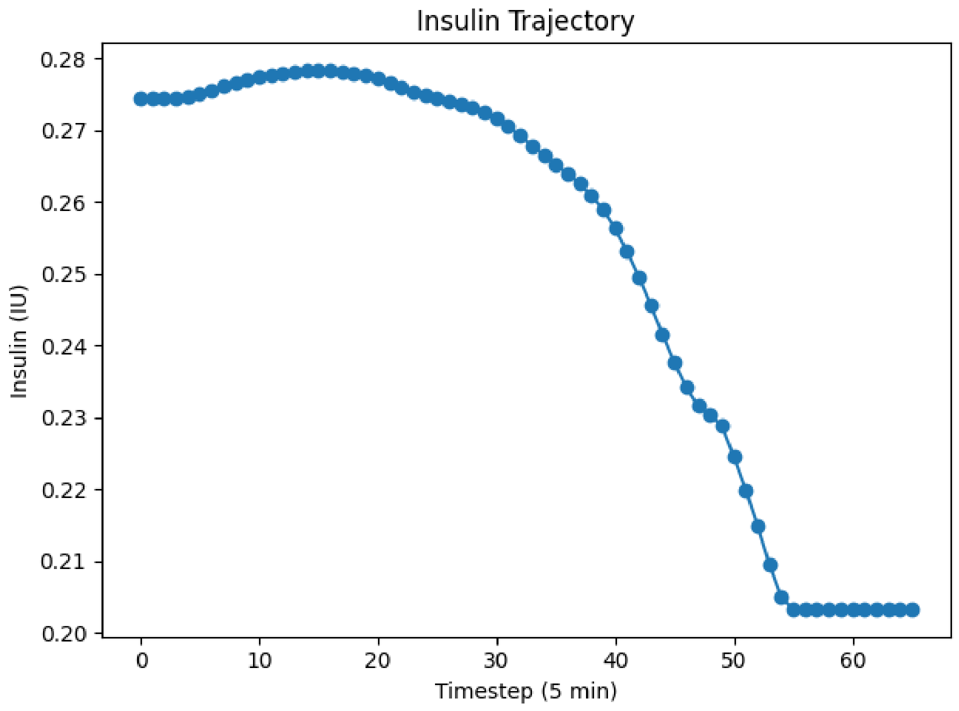
4) PPO with LGBM
→ The RL model does not predict future BG.
→ Using LightGBM as the BG predictor with the Kaggle 1st solution.

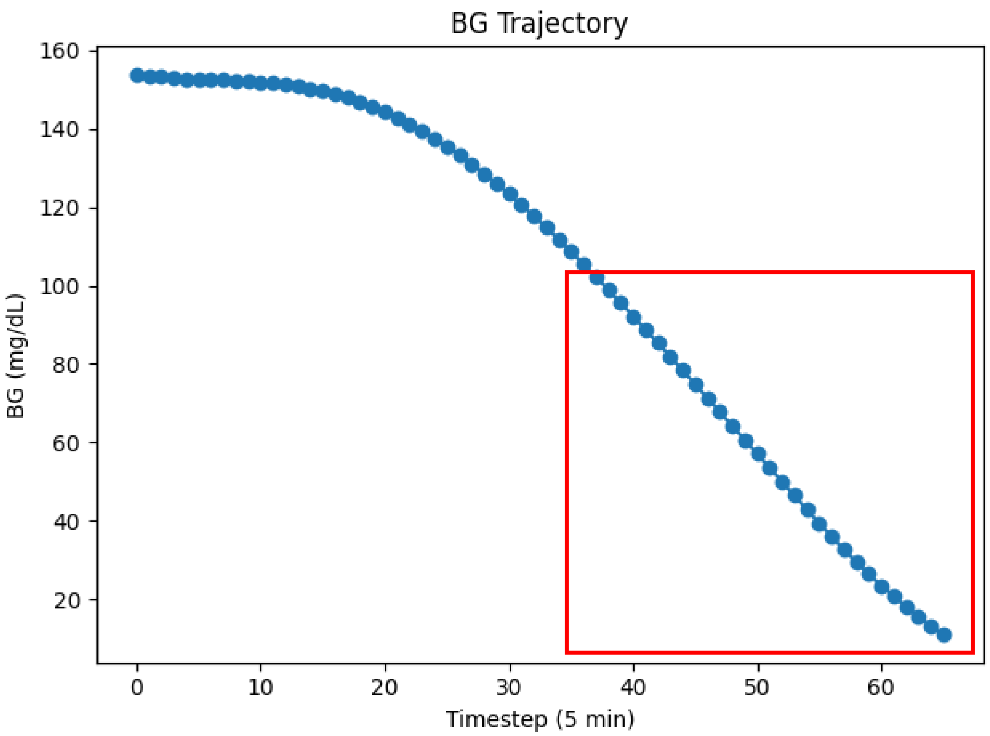




3. Summary of Models.
3.1. SAC with Filter, PPO with LGBM.

4) PPO with LGBM
→ Insulin administration is more delicate.
→ It could replace Filter in SAC, but still has 'low BG control' problem (High LGBI).





3. Summary of Models.
3.1. SAC with Filter, PPO with LGBM.

5) Result Summary
- Applying Filter in SAC, it improved TIR from 23.30 → 61.09 and HBGI 33.12 → 1.94.
- Applying LGBM in PPO, it improved TIR from 22.72 → 60.76 and HGBI 33.39 → 2.73.
- Applying Custom Reward, it improved TIR from 60.76 → 67.93 and LBGI from 41.45 → 28.39.

Method	Time-in-Range (%)	LBGI (mean)	HBGI (mean)
SAC	23.30	0.00	33.12
SAC + Filter	**61.09**	52.39	**1.94**
PPO	22.72	0.00	33.39
PPO + LGBM	**60.76**	41.45	**2.73**
PPO + LGBM + Custom Reward	**67.93**	28.39	**4.61**

3. Summary of Models.

3.2. PPO with LGBM High LBGI.

- To reduce high LBGI in PPO with LGBM, explored a new reward function.
- After apply step function and cliping, the model seemed to contorl BG, but the problem reoccurs again.

→ It's because LGBM is trained to predict after 1 hour of BG,

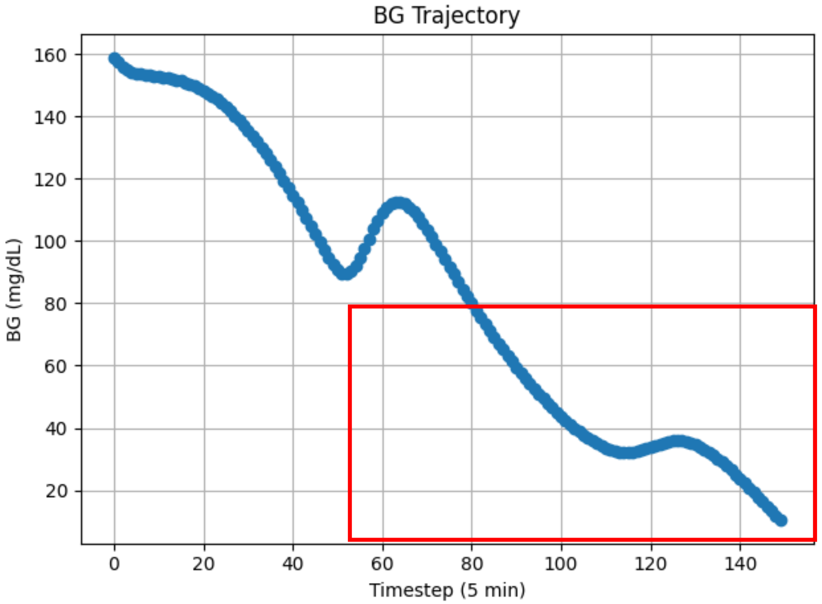
→ but the RL agent might make a mis-trajectory of BG.

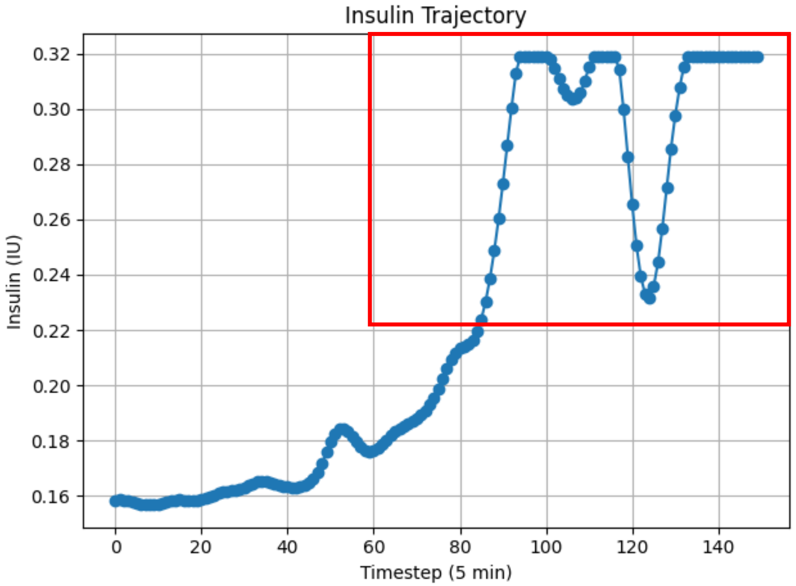
→ So, the simulator BG value and predicted BG value are very different for big timestep.

```python
# 보상 계산: gaussian + severe penalty + delta 보너스
reward = self.gaussian_reward(pred_bg)
# 극단적 hypo/hype 페널티
if pred_bg < 70:
    reward -= (1.0 + 0.05 * (70 - pred_bg))
elif pred_bg > 180:
    reward -= (1.0 + 0.02 * (pred_bg - 180))

# 상승 보너스
delta = pred_bg - current_bg
if current_bg < 80 and delta > 0:
    reward += 0.2 * (delta / 10.0)

new_reward = np.clip(reward, -1.0, 1.0)
return obs, new_reward, terminated, truncated, info
```





4. Benchmark Study.
Dual PPO[1] (2025, SOTA), it is very good compared with our study
→ But it improved PPO, and our study is orthogonal to it.
e.g.) Using Beta Distribution Policy. (Our study uses MLP Policy with Gaussian, as default.)

Time-in-Range (%)	LBGI (mean)	HBGI (mean)
87.45	0	5

5. Discussion.
- The filtering method using Rule Base has limitations.
- The performance is better even if just attach Tree-Model.
- However, 'low BG control' problem is still necessary.

6. Future Work.
- Pre-Train LGBM using simulation data.
- Attempt using Dual PPO model. (Cannot be done because the source code is not disclosed.)

[1] https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0317662#:~:text=blood%20glucose%20regulation,time%20in%20range%20%28TIR

Summary Our Works

1. BG Prediction (Midterm)
→ LightGBM Tree-Based Model

2. BG Control (Final)
→ SAC RL Model with Filter
→ PPO RL Model with LightGBM Predictor

Appendix A. Dual PPO

1. **Policy Entropy Bonus Adjustment**
   - The entropy coefficient is increased to strengthen initial exploration, and then gradually reduced to induce convergence.
2. **Gradient Clipping**
   - To prevent gradient runaway, we set the maximum norm (max_norm) to ensure stable parameter updates.
3. **State Normalization**
   - By mean-variance normalizing the input CGM values, we reduce instability caused by differences in data range during neural network training.
4. **Advantage Normalization**
   - Scale the episode-wise advantage to have mean 0 and variance 1 to improve learning efficiency.
5. **Reward Scalling**
   - Adjust Clarke risk index-based rewards to an appropriate size to mitigate learning bottlenecks caused by being too small or too large.
6. **Use Beta Policy**
   - Instead of restricting actions to the range [0,1], we use the Beta distribution to naturally clip and smooth out actions near the boundaries.
7. **Learning Rate Decay**
   - Set the initial learning rate high and then decrease it linearly or exponentially to ensure convergence stability.
8. **Value Clipping**
   - Similar to PPO clipping, we also apply clipping ranges to value function updates to suppress unstable updates.
9. **Orthogonal Initialization**
   - Initialize neural network weights as orthonormal matrices to reduce gradient vanishing and runaway problems that occur in the early stage of training.
10. **Mini-batch Shuffling**
    - At each epoch, the mini-batches are randomly shuffled to reduce correlation and mitigate overfitting.