SVILUPPO DI UN APPLICATIVO SOFTWARE PYTHON AI-BASED PER LA PREDIZIONE DELL'USURA DEL PIANO MARTIRE DI MACCHINE CNC PER NESTING

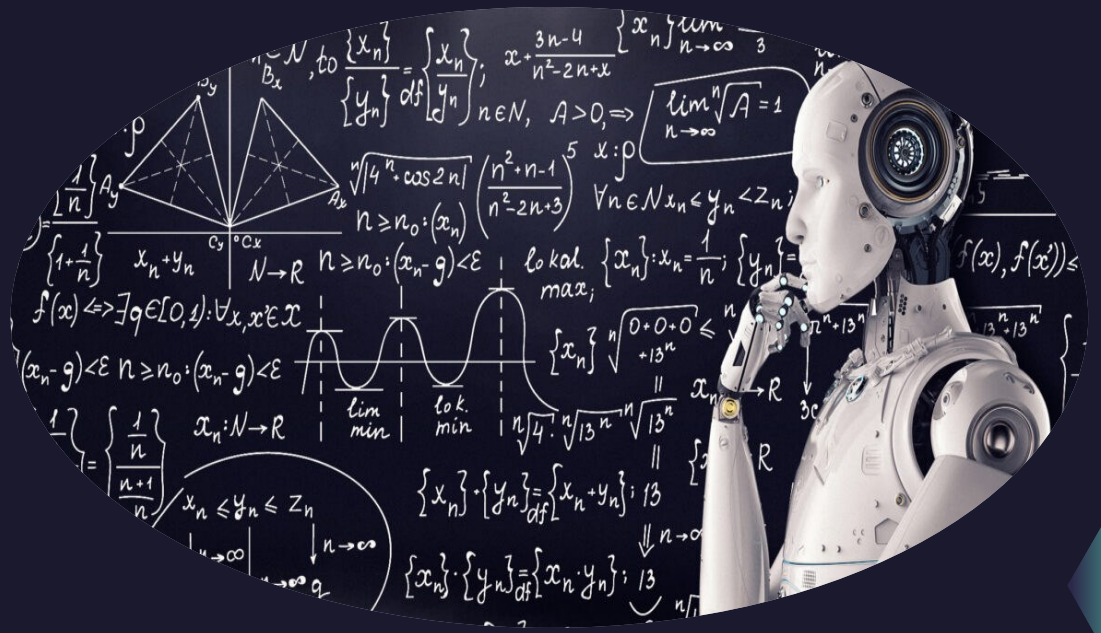
A cura di Filippo Matteini

Introduzione

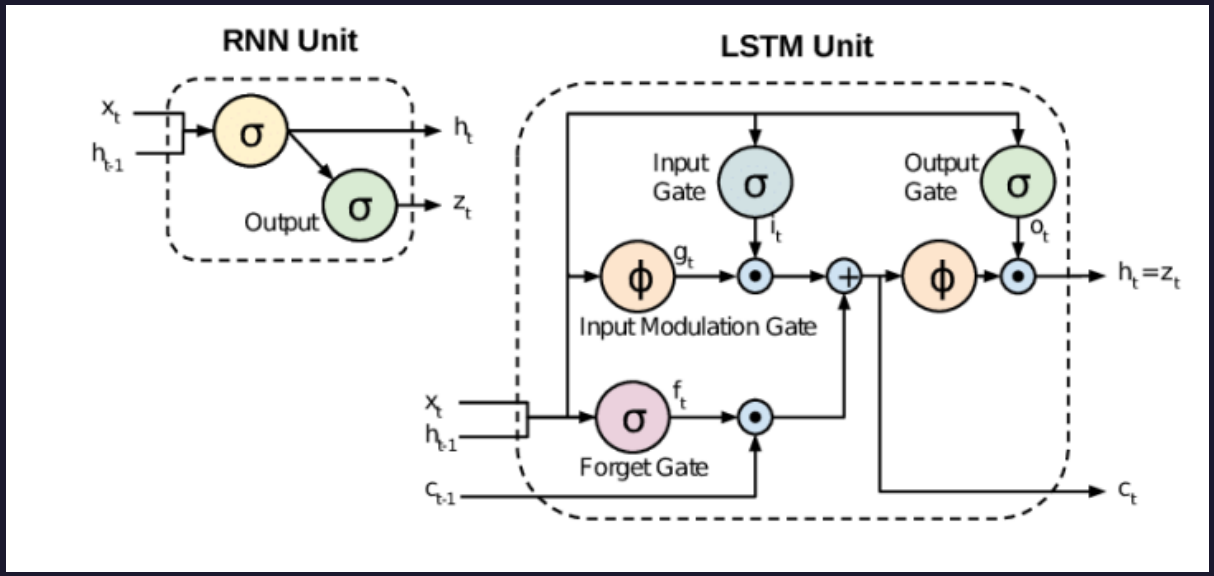
- L'intelligenza artificiale applicata all'ambito industriale è in continua e rapida evoluzione.
- L'applicazione reale di IA - da me sviluppata - è stata quella di implementare un software di manutenzione predittiva per la macchina CNC nesting di SCM Group S.p.A.: **Morbidelli X100-100 Cell**.
- Le macchine CNC nesting di questo tipo sono dotate di un **piano martire:** un piano di lavoro usato come supporto macchina «a perdere» sopra il quale viene effettuata la catena di produzione.
- E' inevitabile che, con l'utilizzo costante della macchina, si sviluppi un'usura progressiva e un deterioramento costante di tale piano di lavoro.
- Come tirocinante ho contribuito alla realizzazione di un applicativo software AI-based per:
    - Diagnostica in tempo reale
    - Ottimizzazione produttiva
    - Previsione operativa e manutentiva del piano martire

Problema generale

- Predizione regressiva in tempo reale dell'andamento produttivo.
- Classificazione del prossimo evento di manutenzione da compiere: Rettifica o Sostituzione.
- Uso del Deep Learning e di reti neurali per ottenere la massima granularità delle predizioni sulle abitudini operative del singolo cliente distinto.



Cenni generali: LSTM vs RNN



- Una Rete Neurale Ricorrente (RNN) è un modello di Deep Learning abile nel lavorare con sequenze di dati contestualizzate.
- LSTM (evoluzione delle RNN), in grado di superare i limiti di memoria e stabilità a lungo termine.
- LSTM Bidirezionale (da me utilizzato): modello che sfrutta due LSTM contrapposte in serie. Invece di analizzare solo "in avanti", la rete tiene conto anche della sequenza appena processata iterando contemporaneamente anche "all'indietro".
- Maggiore attenzione alla dipendenza temporale tra eventi trascorsi e futuri.

Recupero dei dati

- Qualunque architettura di IA funziona bene solo se la qualità e la quantità dei dati passati in input al modello è soddisfacente.
- Questo aspetto ha richiesto la visione e l'analisi di tutte le dinamiche e dei dati disponibili attualmente.
- …ma ben presto ci siamo accorti che questi dati non erano sufficienti ed esaustivi per condurre un addestramento degno di nota...
- Solo 72 record nella tabella DbSpoilboardMaintenanceHistory sul database di Maestro Active... per addestrare una LSTM Bidirezionale servono decine di migliaia di record!

	Id	Timestamp	MaintenanceType	WearValue
	Fil...	Filtro	Filtro	Filtro
49	49	2025-02-06 11:13:52.932964	1	0.0
50	50	2025-02-06 11:44:30.5524438	0	0.75
51	51	2025-02-06 13:54:59.4464332	1	0.6217
52	52	2025-02-06 14:08:17.0416054	1	0.4135
53	53	2025-02-06 14:30:10.1129476	1	0.3215
54	54	2025-02-06 14:37:43.2307798	0	0.75
55	55	2025-02-12 15:03:08.5322316	1	0.2766
56	56	2025-02-13 09:16:20.3767947	1	0.6316
57	57	2025-02-13 09:20:14.5954925	1	0.5627
58	58	2025-02-13 09:21:49.1681642	0	0.75
59	59	2025-02-13 09:27:59.2447093	1	0.5049
60	60	2025-02-13 09:32:48.1832461	1	0.4135
61	61	2025-02-13 09:39:37.802363	0	0.75
62	62	2025-02-17 10:52:54.6785106	1	0.568
63	63	2025-02-17 11:01:11.8180888	1	0.4926
64	64	2025-02-17 11:04:43.3904684	0	0.75
65	65	2025-02-17 13:50:44.8333115	1	0.6275
66	66	2025-02-17 13:59:14.6545814	0	0.5
67	67	2025-02-19 16:30:47.0932789	1	0.6056
68	68	2025-02-19 16:31:18.4696892	1	0.1776
69	69	2025-02-19 17:20:04.7637501	1	0.1776
70	70	2025-02-19 17:20:41.3943076	1	0.1776
71	71	2025-02-20 11:09:21.7896373	1	0.1776
72	72	2025-02-20 17:26:13.4103224	0	0.0

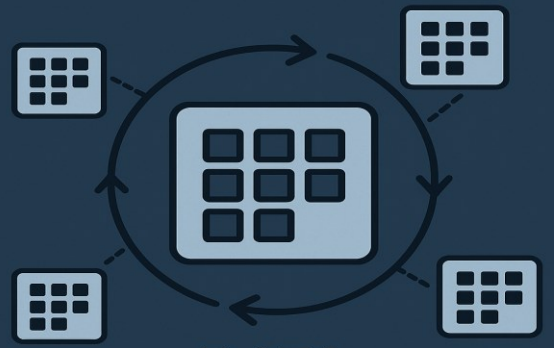
Creazione di dati simulati

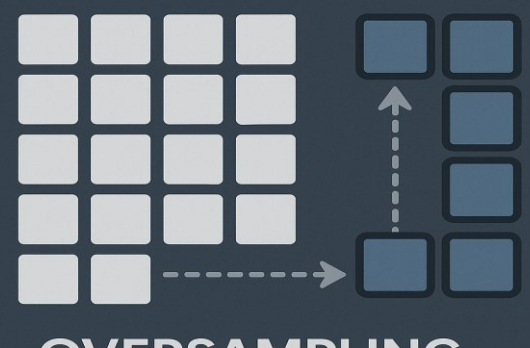
```
Dati simulati - Tail:
                     Timestamp                       Start  \
501961 2022-05-26 08:48:43.593340  2022-05-26 08:48:43.593340
501962 2022-05-26 08:54:44.386745  2022-05-26 08:54:44.386745
501963 2022-05-26 08:56:15.841864  2022-05-26 08:56:15.841864
501964 2022-05-26 09:09:01.811503  2022-05-26 09:09:01.811503
501965 2022-05-26 09:16:59.951818                          0

                             End  EventType  MaintenanceType  WearValue  \
501961  2022-05-26 08:48:51.710756          1               -1      0.000
501962  2022-05-26 08:54:51.830449          1               -1      0.000
501963  2022-05-26 08:56:19.156405          1               -1      0.000
501964  2022-05-26 09:09:03.729087          1               -1      0.000
501965                           0          0                1      0.705

       WorkingHours  TotalEvents
501961         0.00            1
501962         0.00            1
501963         0.00            1
501964         0.00            1
501965        55.22          451
```

- Ho creato una funzione di popolamento di un dataset simulato con dati fittizzi sulla base di quelli reali.
- I campi fondamentali per le manutenzioni sono:
  - "WearValue"
  - "MaintenanceType"
  - "Timestamp"
- I campi fondamentali per le produzioni sono:
  - "Start"
  - "End"
- "Timestamp" è il campo utilizzato dal datetime delle manutenzioni e come key per l'ordinamento del dataset.

Pre-processing

- Pulizia dei dati (valori mancanti, normalizzazione, standardizzazione, codifica ecc).
- Creazione di contesto, con features engineering, per potenziare il legame semantico tra i dati principali.
- Divisione del dataset in features e target.
- Data augmentation on-the-fly per arricchire sinteticamente le features già presenti con esempi derivati.
- Oversampling e split del dataset on-the-fly – tramite pipeline tf.generator - con stratificazione per bilanciare le classi degli eventi manutentivi (Rettifica e Sostituzione) nei train, val e test set.
- Creazione delle sequenze temporali – di input alla rete e generate a partire dal dataset – e zero padding per uniformarle tutte alla stessa lunghezza massima.

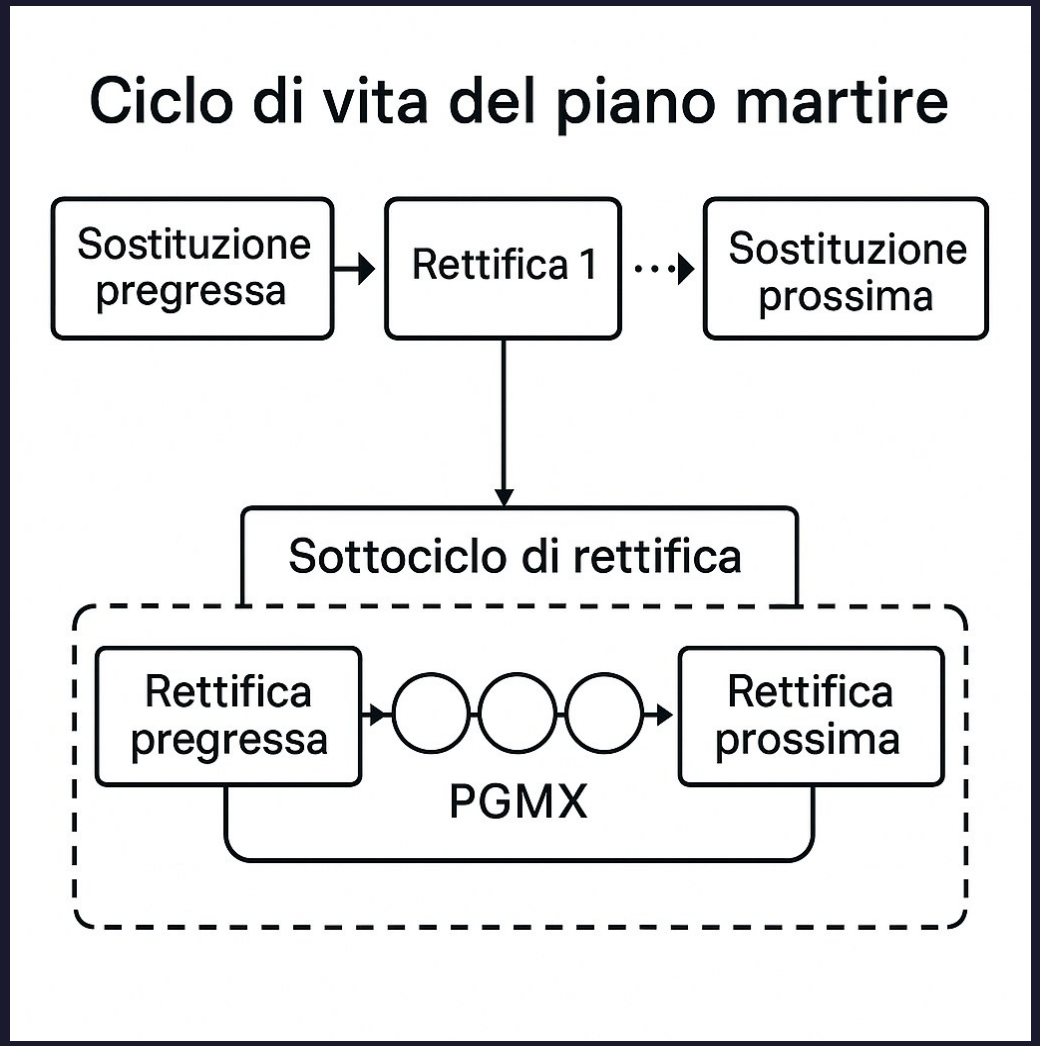




Logica dell'algoritmo

- La logica dell'algoritmo è stata implementata con due cicli innestati:
  - Ciclo di vita del piano martire (tra due sostituzioni consecutive)
  - Sottociclo di manutenzione (tra due manutenzioni consecutive).
- L'unità computazionale unitaria dell'algoritmo è il singolo "pgmx“: ovvero un programma di lavorazione.
- L'unità di misura, per la rete, è il ciclo di vita: tra due sostituzioni consecutive si svolgono N sottocicli e M pgmx con distribuzione:

$$Lavorazioni\ per\ ciclo = \sum_{j=1}^{N} M_j$$



Clienti di default: 3 dataset di base

- L'obiettivo della rete neurale è adattarsi alle abitudini operative di ogni cliente distinto, in un secondo momento, tramite il processo di fine-tuning.

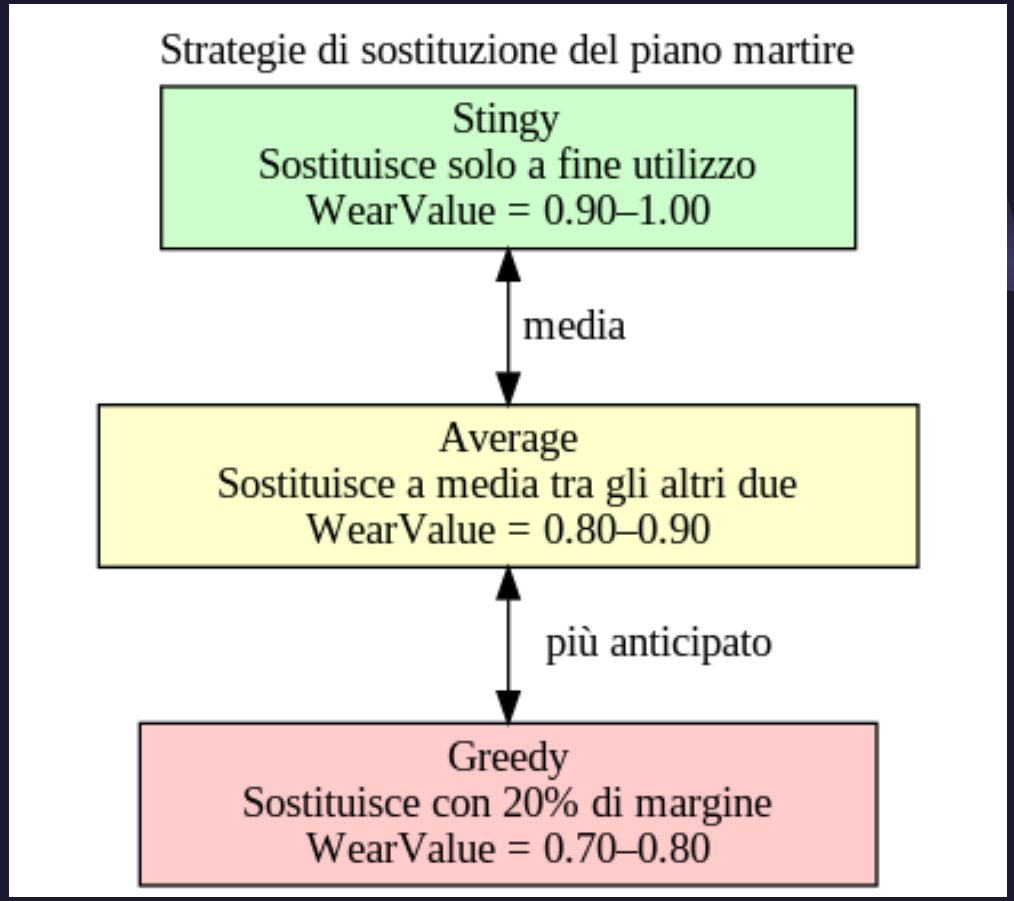
- Tuttavia, per l'addestramento profondo iniziale, ho ideato tre casi di dataset di default con cui allenare il modello, i quali rispecchiano le macro-abitudini possibili e adottabili dai clienti:

  - **Stingy**: sfrutta il piano martire fino alla fine.

  - **Greedy**: tende a fare manutenzione quando il piano è ancora leggermente utilizzabile.

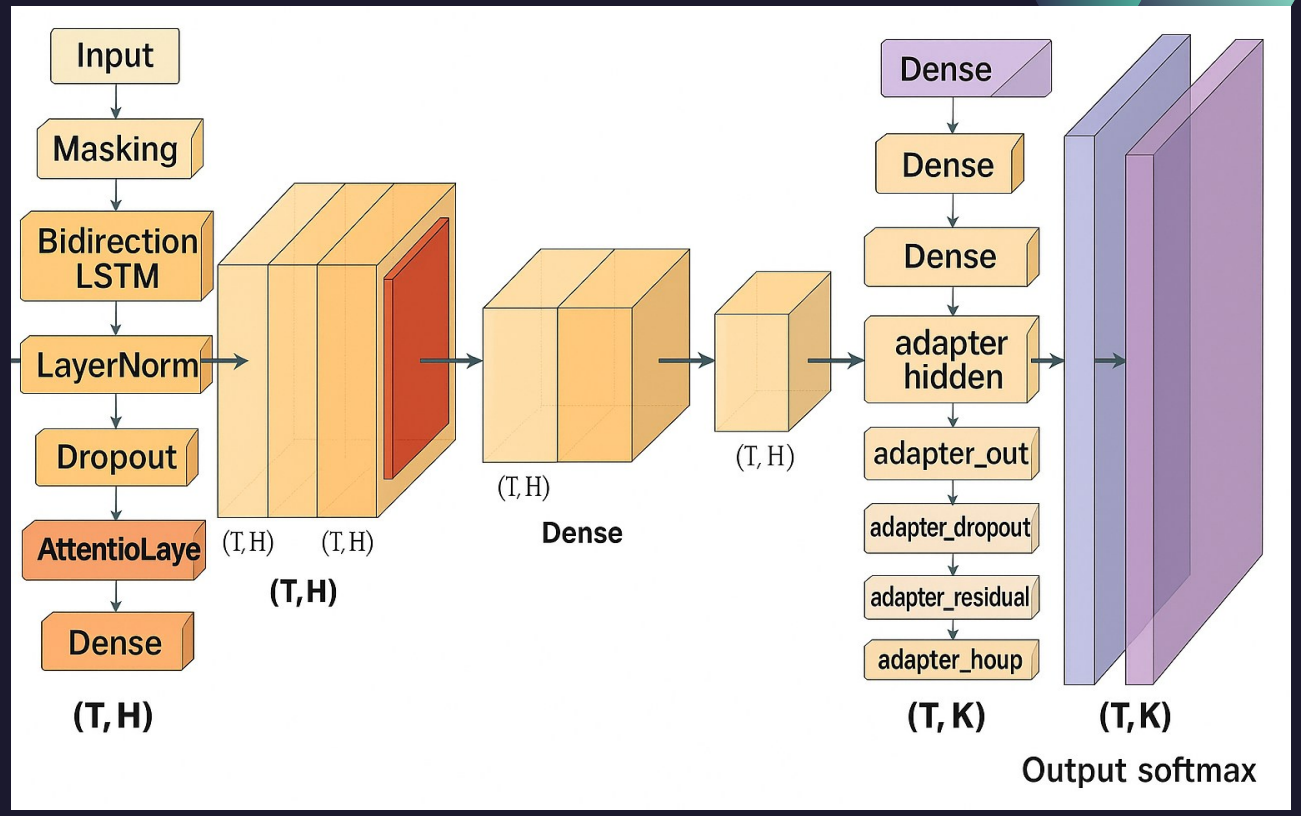
  - **Average**: è una media tra le due strategie precedenti.

- In questo modo, il procedimento successivo di fine-tuning sarà - nella maggior parte dei casi - una rifinitura ulteriore dell'addestramento iniziale già avvenuto.



Architettura del modello

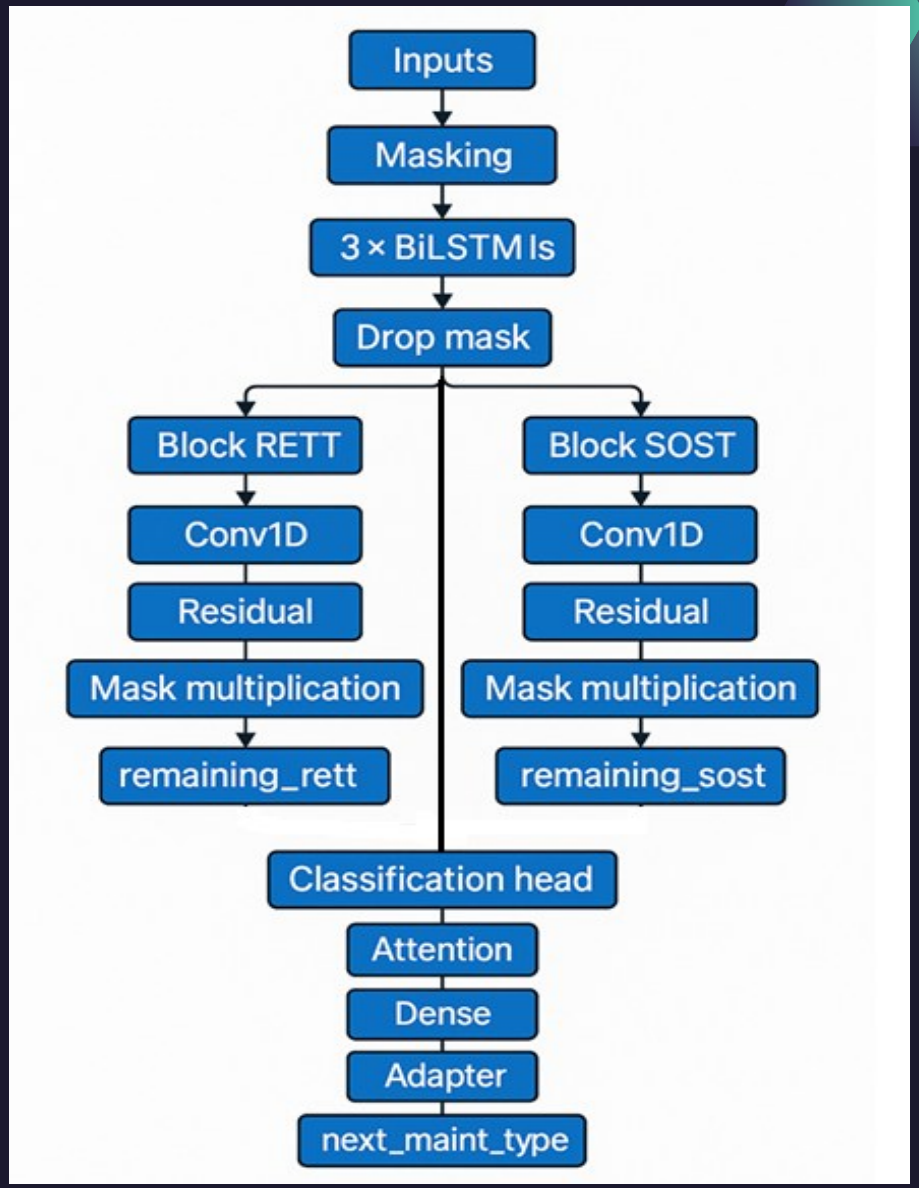
- Nell'immagine adiacente si può osservare l'intera architettura di Deep Learning che ho progettato.
- Rete neurale ricorrente LSTM Bidirezionale multi-output con meccanismo di Self-Attention e un layer Dense di Adapter.
- Self-Attention permette alla rete di analizzare le sequenze in parallelo, dando importanza maggiore al contesto e non solo ad input/output sequenziale.



- Adapter layer per non incorrere in "catastrophing forgetting" (dimenticanza totale) e favorire il mantenimento della conoscenza acquisita durante il processo di fine-tuning.

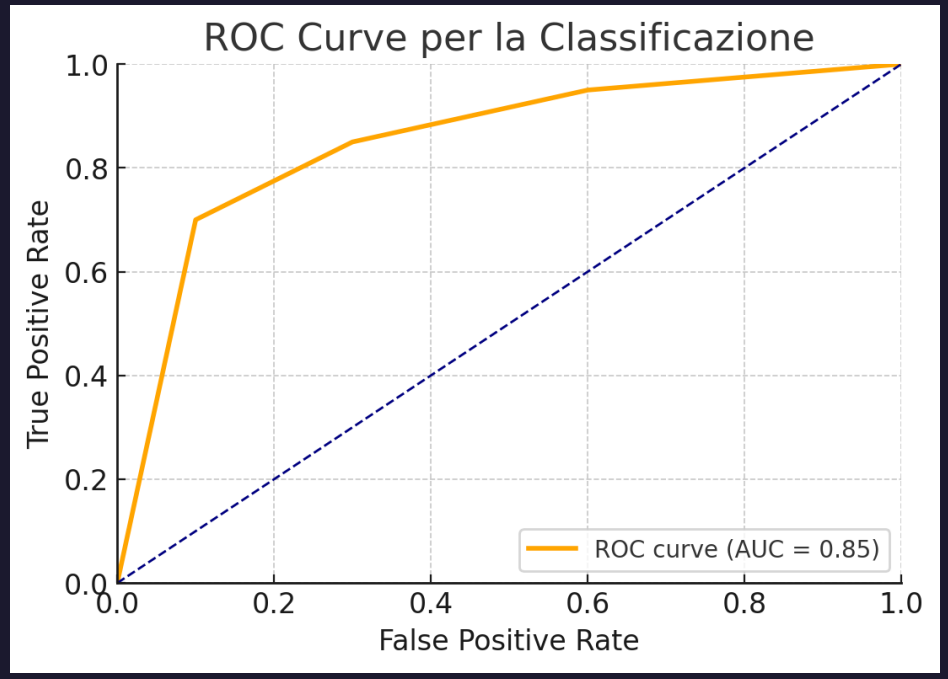
Architettura del modello

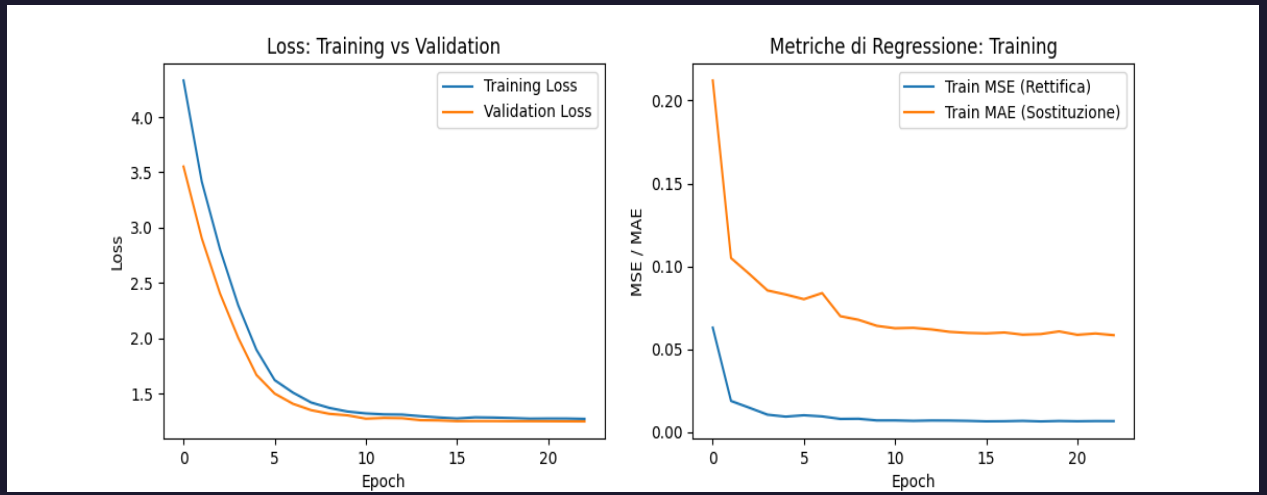
- Il modello è composto da tre layer Bidirectional LSTM, che costituiscono il backbone in comune della regressione e classificazione.
- A partire da essi, si indirizza la rete nei due branch principali per i due compiti distinti.
- Il branch regressivo viene, a sua volta, suddiviso in altri due sotto-branch, per la rettifica e la sostituzione, con layer Dense finali e funzioni di attivazione "linear".
- Il branch classificativo viene gestito in maniera fully-connected con layer di Adapter innestato e layer Dense finale con attivazione "sigmoid".



Performance del modello

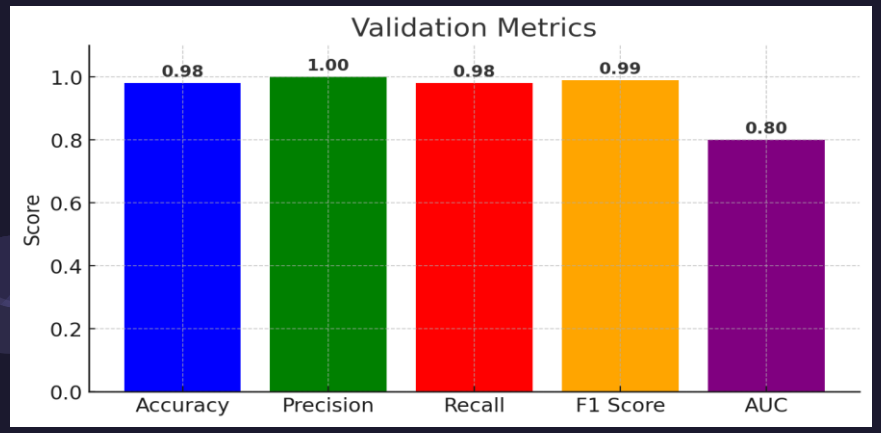
- Le performance del modello possono essere misurate grazie alle metriche di validazione.
- Le metriche che ci interessano in un problema di manutenzione predittiva regressivo e classificativo sono:
  - ROC-AUC: possibilità del modello di distinguere correttamente le due classi.
  - MAE: errore assoluto medio dei valori regressivi lungo i timestep.
  - MSE: errore quadratico medio dei valori regressivi lungo i timestep.





Performance del modello

- A fianco, un piccolo screenshot – preso da terminale - dell'ultima epoca di addestramento.
- Una ROC-AUC raggiunta di circa 80% in validazione.
- La loss classificativa continua a decrescere e questo è un buon segnale.
- Lo stesso avviene per le loss regressive.



Epoch 8 ROC-AUC: 0.8124

247/247 ━━━━━━━━━━━━━━━━━━━━ 177s 5s/step - loss: 1.4285 - next_maint_type_accuracy: 0.9065 - next_maint_type_auc: 0.8691 - next_maint_type_f1_score_metric: 0.9353 - next_maint_type_loss: 0.0718 - next_maint_type_precision: 0.9090 - next_maint_type_recall: 0.9633 - remaining_rett_loss: 0.0192 - remaining_rett_mae: 0.1555 - remaining_rett_mse: 0.0389 - remaining_sost_loss: 0.0447 - remaining_sost_mae: 0.2508 - remaining_sost_mse: 0.1004 -

Validation set:

val_loss: 1.2939 - val_next_maint_type_accuracy: 0.9700 - val_next_maint_type_auc: 0.8051 - val_next_maint_type_f1_score_metric: 0.9880 - val_next_maint_type_loss: 0.0623 - val_next_maint_type_precision: 0.9700 - val_next_maint_type_recall: 1.0000 - val_remaining_rett_loss: 0.0101 - val_remaining_rett_mae: 0.1081 - val_remaining_rett_mse: 0.0240 - val_remaining_sost_loss: 0.0455 - val_remaining_sost_mae: 0.2720 - val_remaining_sost_mse: 0.1251 - learning_rate: 0.0010

Epoch 8: early stopping - Restoring model weights from the end of the best epoch: 6.

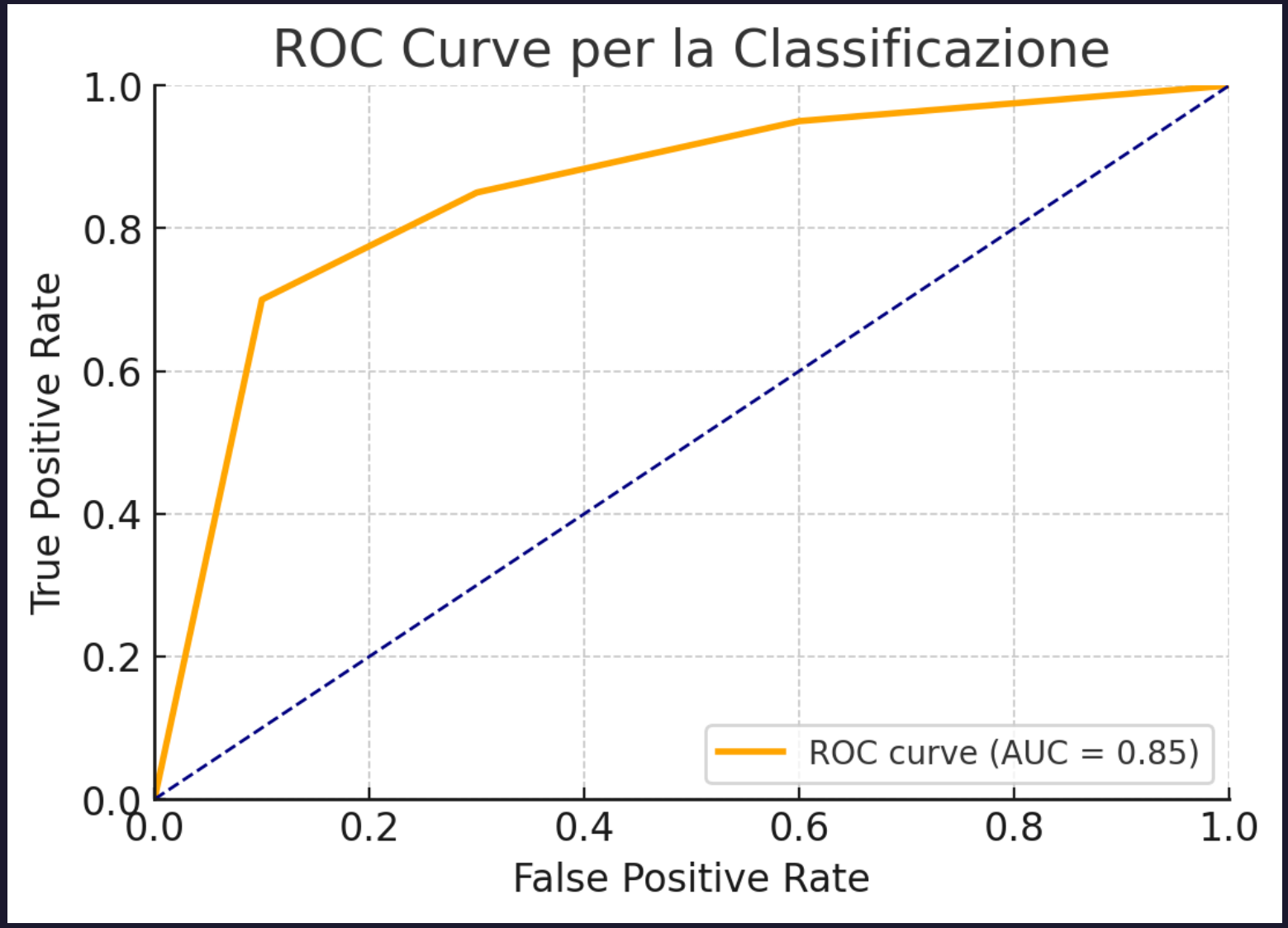
1/4 ━━━━━━━━━━━━━━━━━━━━ 11s 4s/step - loss: 1.9204 - next_maint_type_accuracy: 1.0000 - next_maint_type_auc: 0.8255208134651184 - next_maint_type_f1_score_metric: 1.0000 - next_maint_type_loss: 0.0337 - next_maint_type_precision: 1.0000 - next_maint_type_recall: 1.0000 - remaining_rett_loss: 0.0322 - remaining_rett_mae: 0.2274 - remaining_rett_mse: 0.0737

2/4 ━━━━━━━━━━━━━━━━━━━━ 2s 1s/step - loss: 1.9238 - next_maint_type_accuracy: 1.0000 - next_maint_type_auc: 0.8323785628368 - next_maint_type_f1_score_metric: 1.0000 - next_maint_type_loss: 0.0336 - next_maint_type_precision: 1.0000 - next_maint_type_recall: 1.0000 - remaining_rett_loss: 0.0323 - remaining_rett_mae: 0.2278 - remaining_rett_mse: 0.0739 - remainin

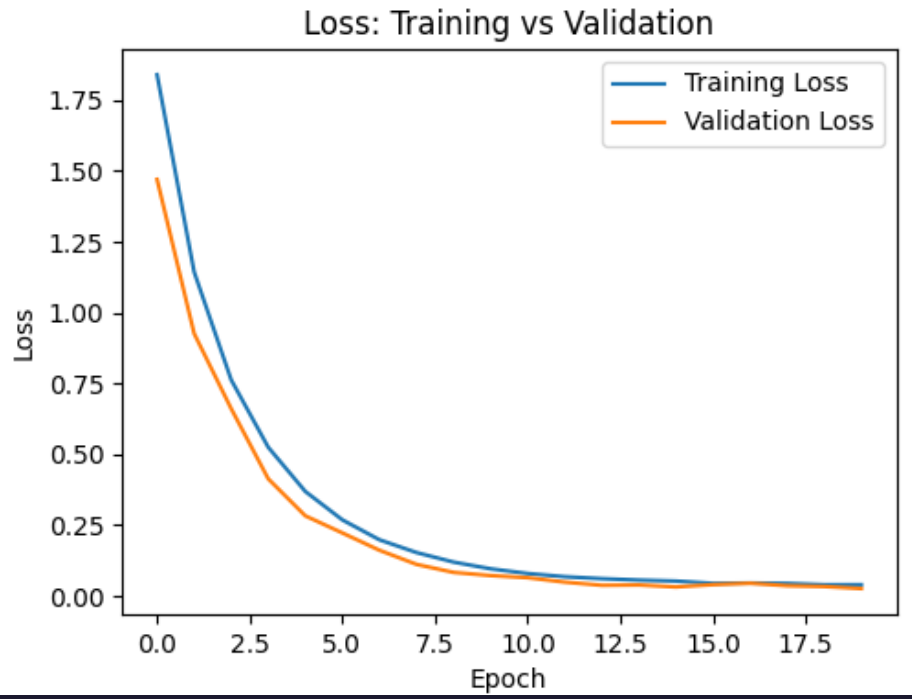
3/4 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - loss: 1.9268 - next_maint_type_accuracy: 0.9896 - next_maint_type_auc: 0.8406 - next_maint_type_f1_score_metric: 0.9945 - next_maint_type_loss: 0.0343 - next_maint_type_precision: 0.9896 - next_maint_type_recall: 1.0000 - remaining_rett_loss: 0.0323 - remaining_rett_mae: 0.2278 - remaining_rett_mse: 0.0739

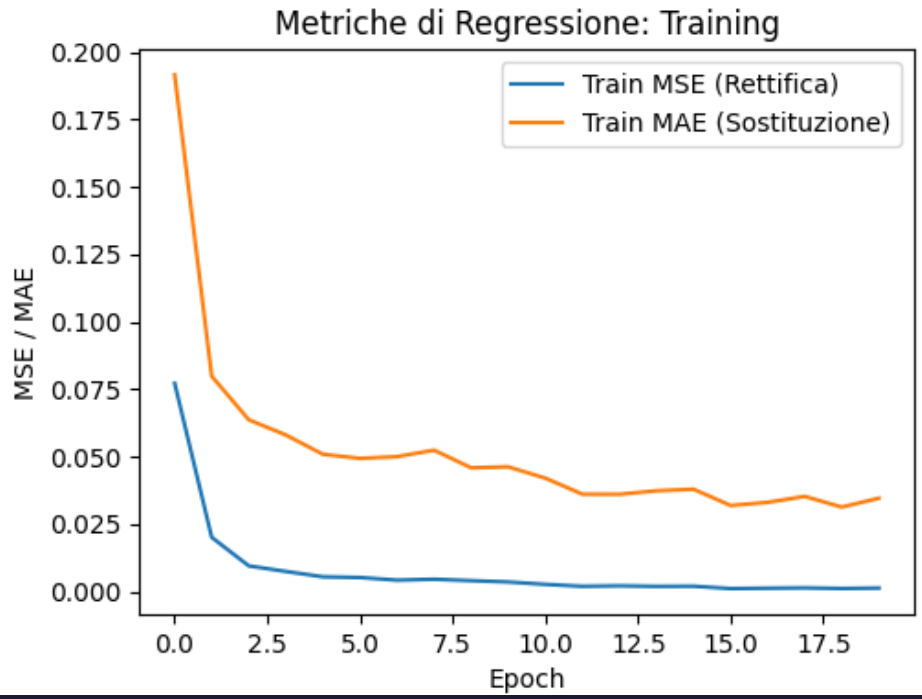
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 833ms/step - loss: 1.9244 - next_maint_type_accuracy: 0.9778 - next_maint_type_auc: 0.8668 - next_maint_type_f1_score_metric: 0.9775 - next_maint_type_loss: 0.0364 - next_maint_type_precision: 0.9778 - next_maint_type_recall: 1.0000 - remaining_rett_loss: 0.0323 - remaining_rett_mae: 0.2279 - remaining_rett_mse: 0.0739 - remaining_sost_loss: 0.0553 - remaining_sost_mae: 0.3264 - remaining_sost_mse: 0.1670

Un po' di grafici...

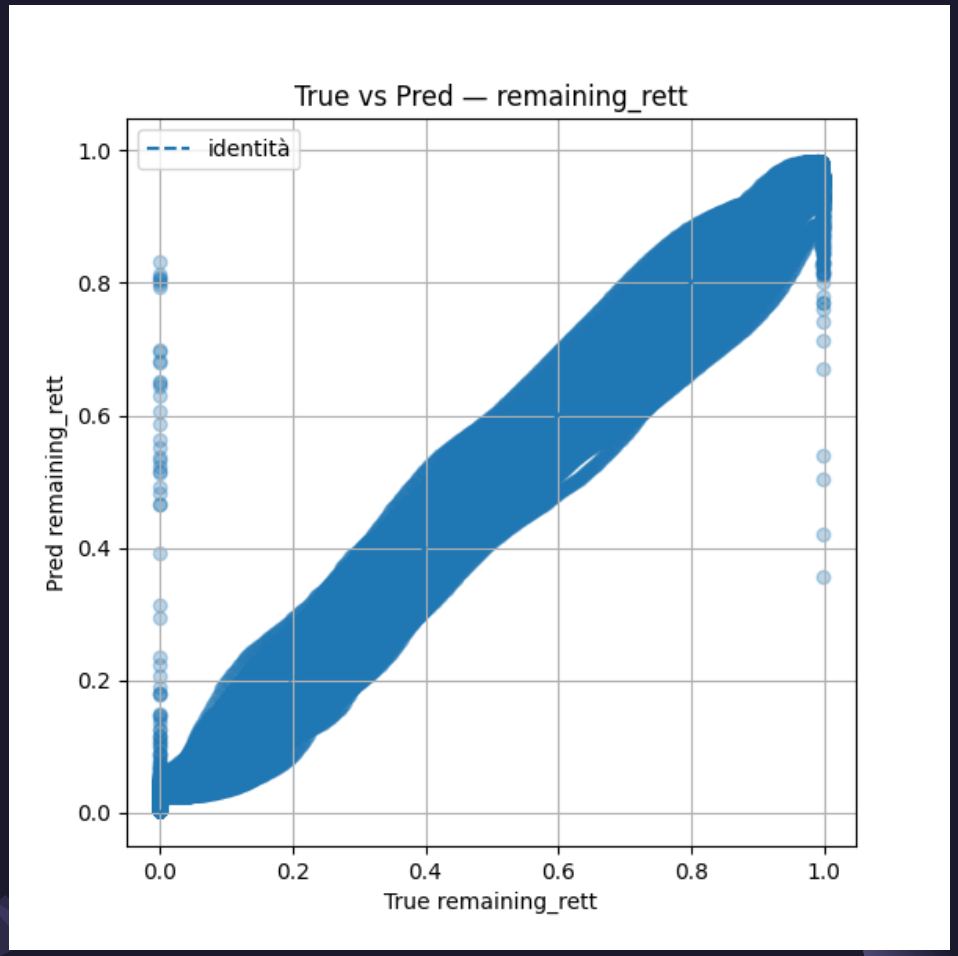


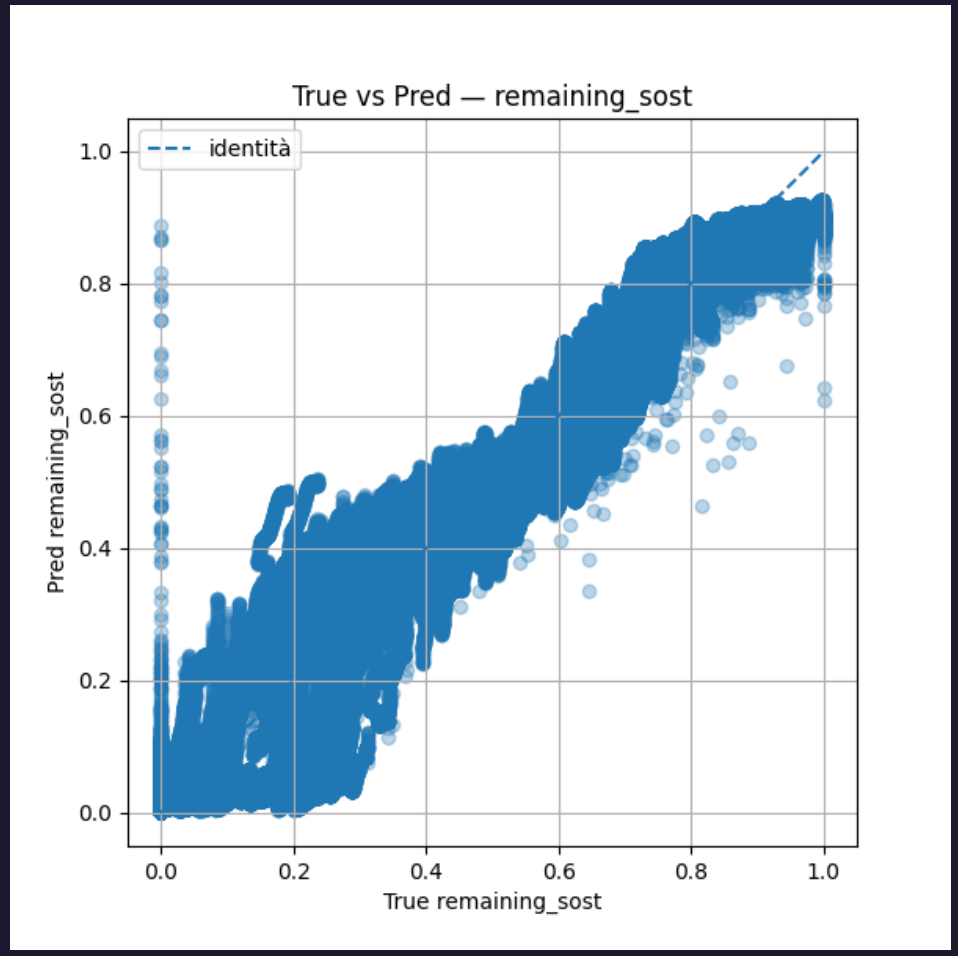
Un po' di grafici...





Un po' di grafici...





```python
# congela / sblocca
# Freezo tutto
for l in model.layers:
    l.trainable = False
# e sblocco adapter + testa di classificazione
for name in ("adapter_hidden", "adapter_out", "adapter_residual", "adapter_dropout", "adapter_ln", "next_maint_type"):
    model.get_layer(name).trainable = True
if strategy == "hybrid":
    for l in model.layers:
        if isinstance(l, tf.keras.layers.LayerNormalization):
            l.trainable = True

# congela / sblocca e ricompila
ensure_compile(model, base_lr, mode="full")
model.optimizer.learning_rate.assign(base_lr)
```

Transfer Learning

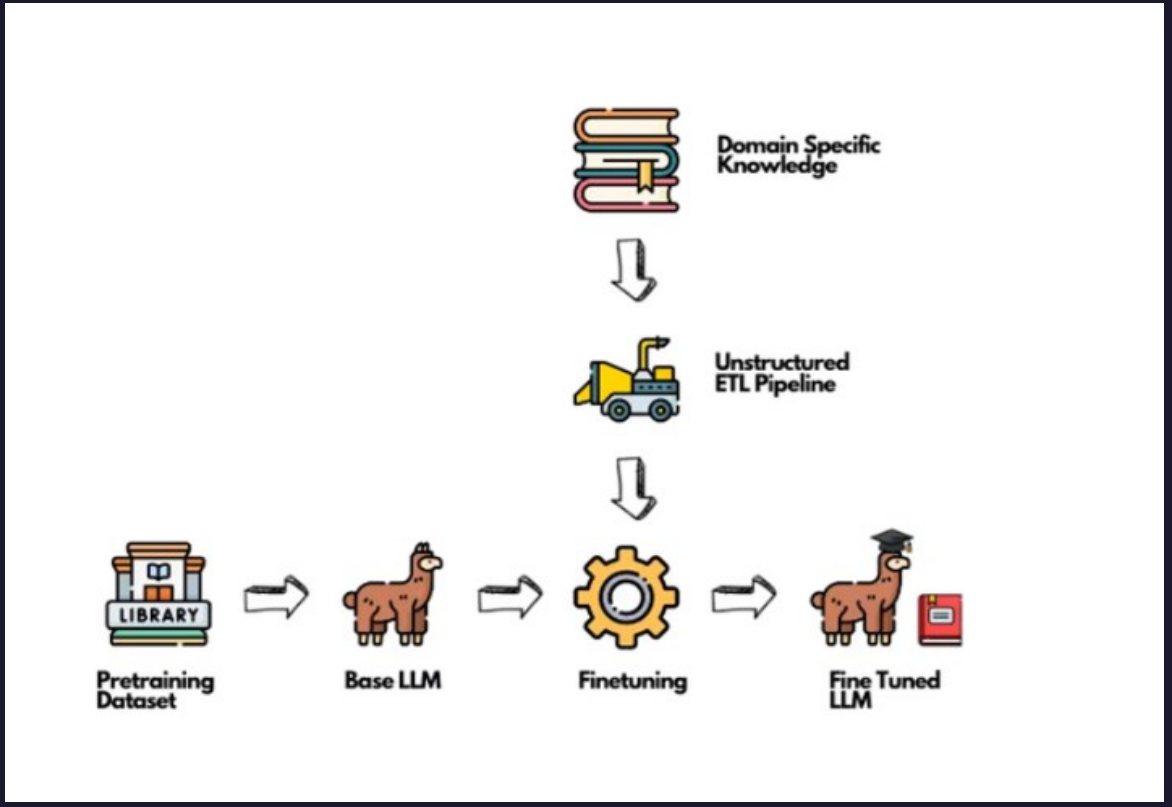
- Per garantire il trasferimento e mantenimento di conoscenza risultante dall'addestramento di default, è necessario applicare un congelamento di alcuni strati della rete.
- Si congelano alcuni livelli (di solito i primi) e si ri-addestrano le teste finali sui nuovi dati che arrivano dalla produzione.
- In questo modo, si favorisce l'adattamento ad ogni cliente specifico senza però perdere le conoscenze acquisite inizialmente e incorrere nel "catastrophing forgetting".
- Tenere ri-addestrabile l'adapter serve a rendere più flessibile il modello ai nuovi dati.
- "Step classification flush" vs "Step regression-only" con strategia "hybrid".

Fine-tuning

- E' il vero nucleo del meccanismo di apprendimento e adattamento della rete in deployment.
- Si ricalibrano i pesi del modello ad ogni iterazione di fine-tuning e si aggiornano dinamicamente le predizioni aggiustandole con un re-fitting rapido.
- Nel mio caso, ho attuato due processi distinti di fine-tuning, "regression-only" e "full":

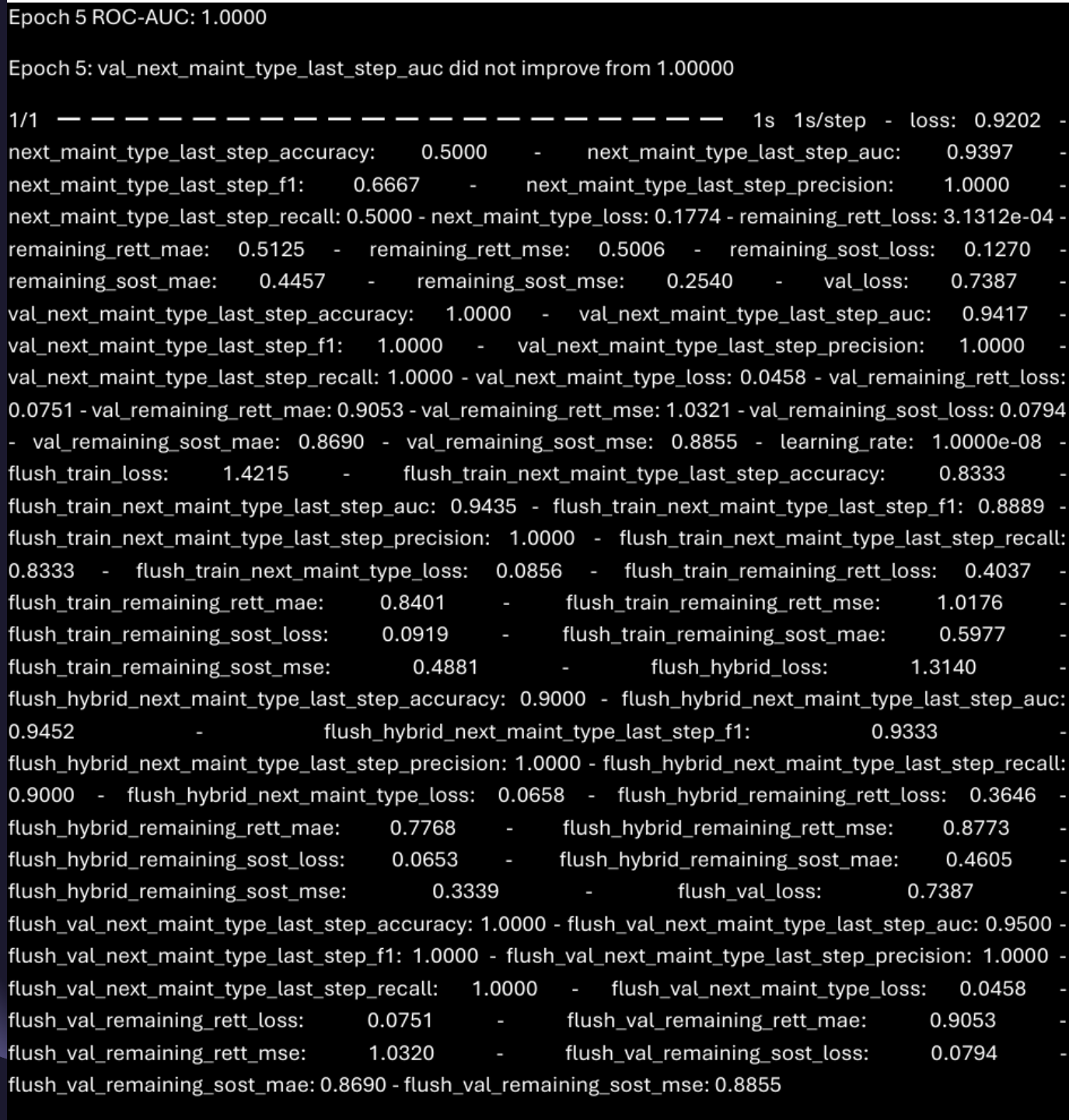
  - "regression-only": ri-addestra velocemente le teste finali dei branch regressivi ad ogni nuova manutenzione registrata nello storico.

  - "full": esegue un ri-addestramento dell'adapter layer e dei LayerNormalization, oltre che di tutte le teste finali di classificazione e regressione, ad ogni nuovo evento di sostituzione registrato.

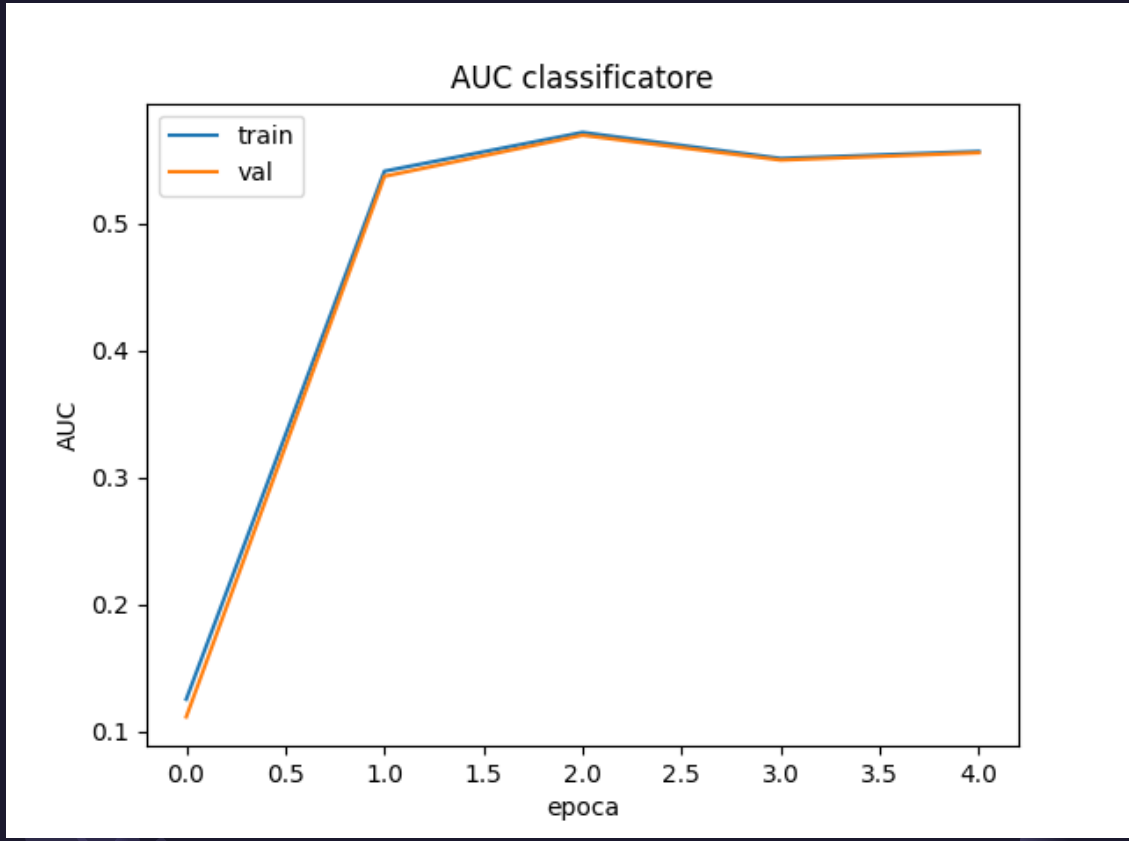


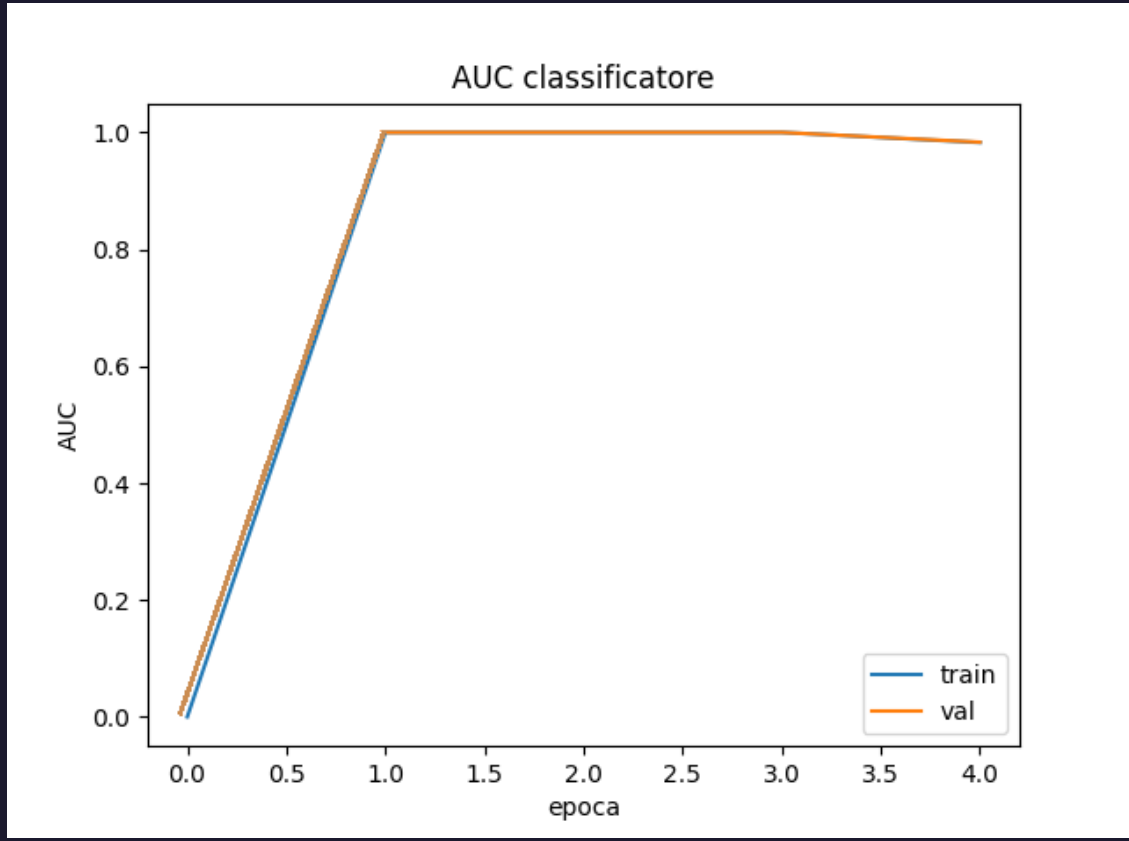
Performance del fine-tuning

- A fianco, un piccolo screenshot – preso da terminale - dell'ultima epoca di fine-tuning.
- Le prestazioni sono rimaste elevate, quindi il transfer learning ha lavorato coerentemente con la problematica affrontata.
- E' stato testato su un numero esiguo di dati aggiuntivi, ma le metriche sono, generalmente, rimaste elevate anche in questa casistica.
- Sarebbe interessante osservare il comportamento anche con un numero di dati più consistente a disposizione.



Un po' di grafici...





Server ReST e deployment

- Per rendere l'applicazione disponibile al cliente, è stato necessario realizzare un servizio ReST con Flask API, per creare un server che andasse a comunicare direttamente con i vari client, tramite gli opportuni end-points:

  - **GET /predictions**: restituisce il JSON con la predizione corrente.

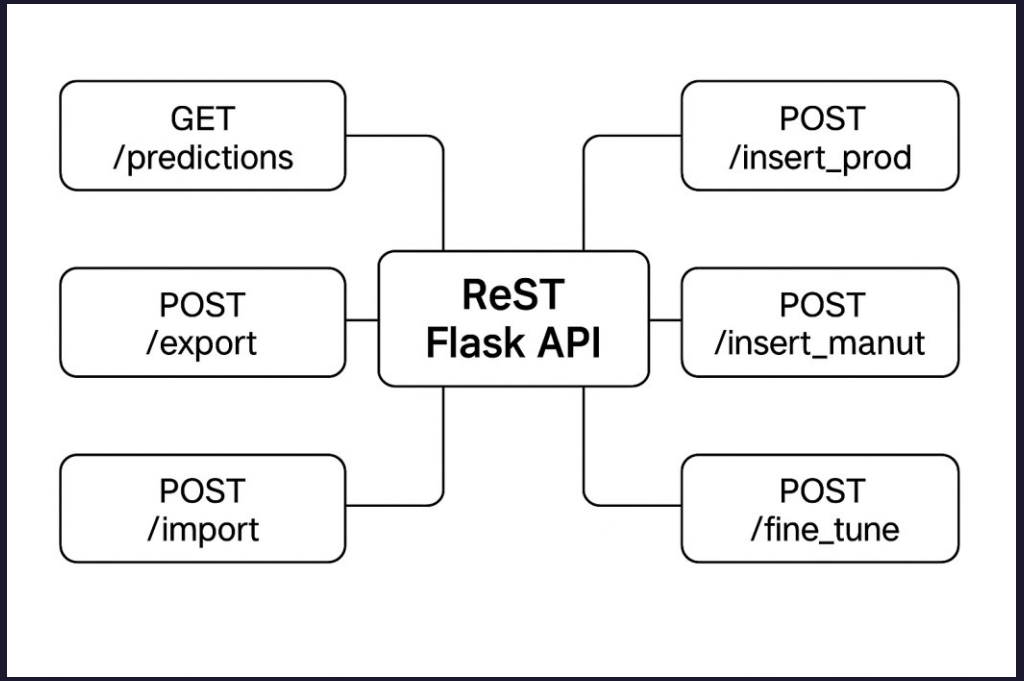
  - **POST /fine_tune**: si occupa di effettuare fine-tuning tra "regression-only" e "full".

  - **POST /import**: importa i pesi di un modello in formato .h5.

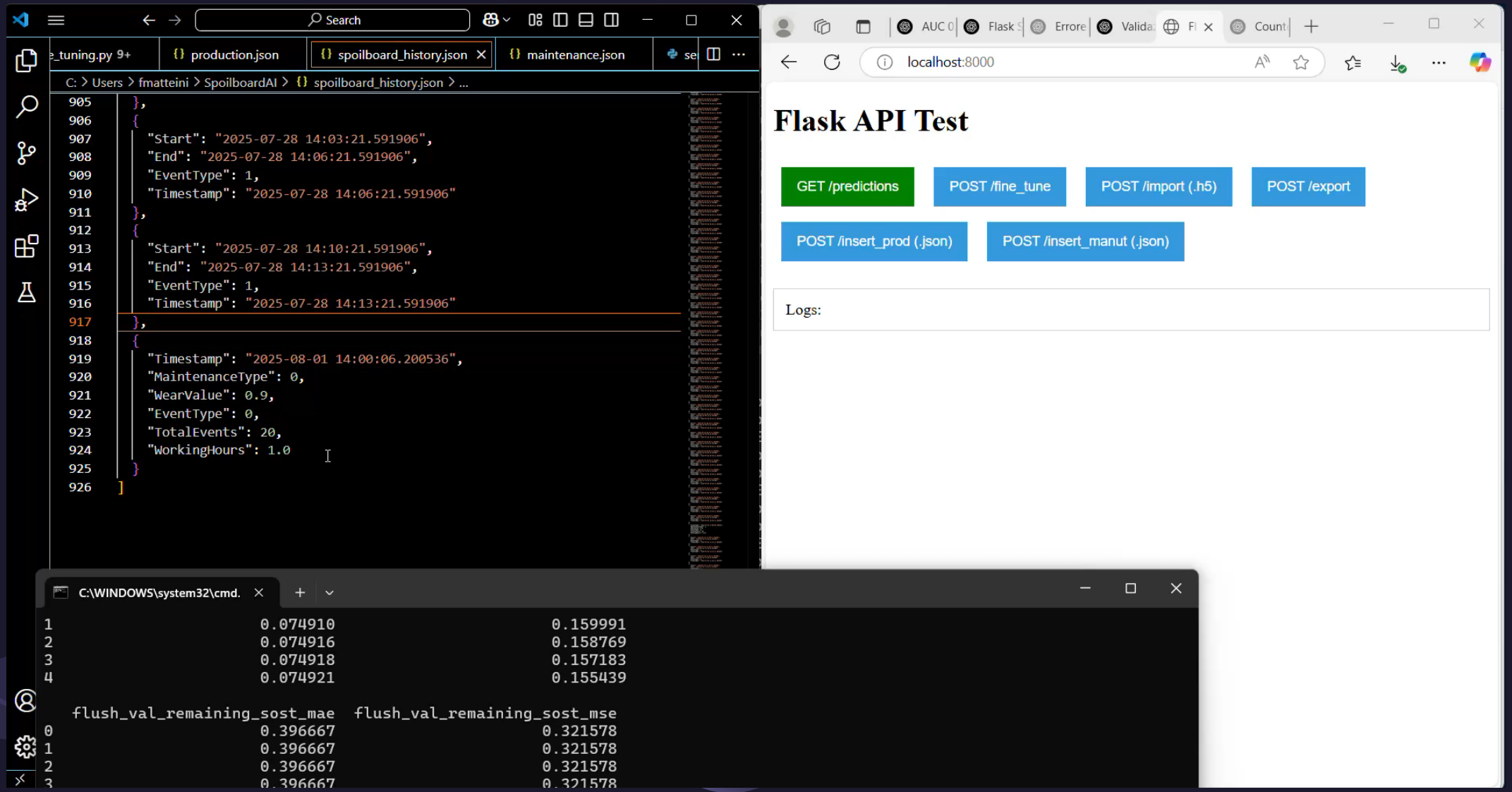
  - **POST /export**: esporta i pesi di un modello in formato .h5.

  - **POST /insert_prod**: inserisce un record JSON di produzione del nuovo storico.

  - **POST /insert_manut**: inserisce un record JSON di manutenzione del nuovo storico.



Demo e simulazione



Sviluppi futuri

- Utilizzo del Reinforcement Learning per implementare un sistema di aggiustamento delle predizioni tramite ACK con feedback da parte dell'utente.
- Applicazione della manutenzione predittiva non solo sul piano martire complessivo, ma anche sulla suddivisione dell'area totale in zone separate, trattandole come un piano martire a sè stante.
- Estensione del concetto di manutenzione predittiva agli utensili e supporti della macchina.
- Migrazione ad un'architettura neurale Transformer su hardware GPU per prestazioni ancora migliori.

Grazie a tutti per l'attenzione!

FINE