

## AI Brain Factory

Este projeto “AI Brain Factory” tem a intenção de criar um editor de modelos de IA - Assistente de edição de modelos distill pré treinados para processo educativo do funcionamento de treinamento com fine tuning e supervisão,

Transformando a educação através de diálogos inteligentes e personalizados\*

O Chat\_HITL\_RL\_v1, iniciou e processou todo fluxo de modo satisfatório apenas com cpu. Uma das premissas era rodar em qualquer computador.

Requisitos de CPU: I7 RAM: 32gb,

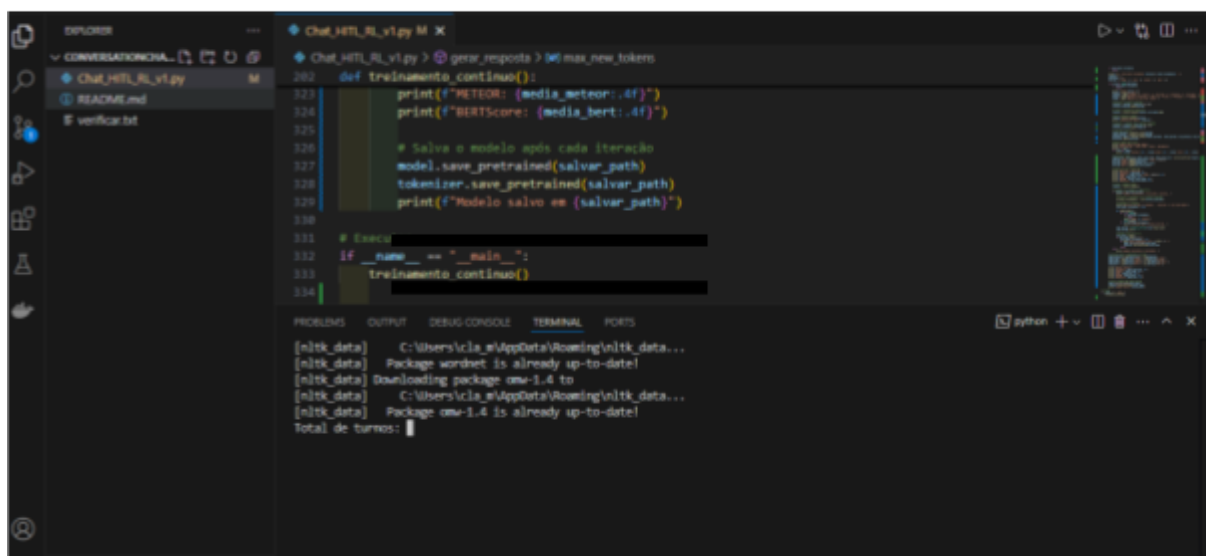
Com menor desempenho foi testado em computadores: CPU: I5 e I3 - RAM 16gb ou CPU: FX6300 RAM: 16gb.

Computadores com RAM menor de 10gb o código encerrou e o computador reiniciou.

O objetivo é mostrar para alunos o passo a passo do desenvolvimento do código e conceitos de treinamento de modelos “distill” pré treinados com técnicas de supervisão e fine tuning.

Não há intenção, ou objetivo, de treinar qualquer modelo neste método, pois a forma correta

Fig.1



```
Chat_HITL_RL_v1.py M x
Chat_HITL_RL_v1.py > @ generate_response > @ max_new_tokens
280 def treinamento_continuo():
281     print(f"METRICA: (media_meteor:.4f)")
282     print(f"BERTScore: (media_bert:.4f)")
283
284     # Salva o modelo após cada iteração
285     model.save_pretrained(salvar_path)
286     tokenizer.save_pretrained(salvar_path)
287     print(f"Modelo salvo em (salvar_path)")
288
289 # Exemplo
290 if __name__ == "__main__":
291     treinamento_continuo()
292
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[nltk_data] C:\Users\icla\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\...
[nltk_data] Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data] C:\Users\icla\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\...
[nltk_data] Package omw-1.4 is already up-to-date!
Total de turnos: 1
```

é com grande volume de dados e com uso de GPU com poder de processamento muito maior..

Sabendo que nem todas pessoas possuem computadores com requisitos para treinar da forma mais eficiente foi idealizado este método, para ensinar conceitos básicos de treinamento de modelos destilados para processamento de linguagem natural.

A figura mostra o momento em que o código é iniciado no terminal, onde o aluno depois de escrever o código pode interagir, pois são solicitadas informações de análise do comportamento do modelo.

São dados básicos para interagir como na Fig.2:

“Total de Turnos:” são quantos turnos vai ser o diálogo entre o aluno e o modelo de diálogo escolhido.

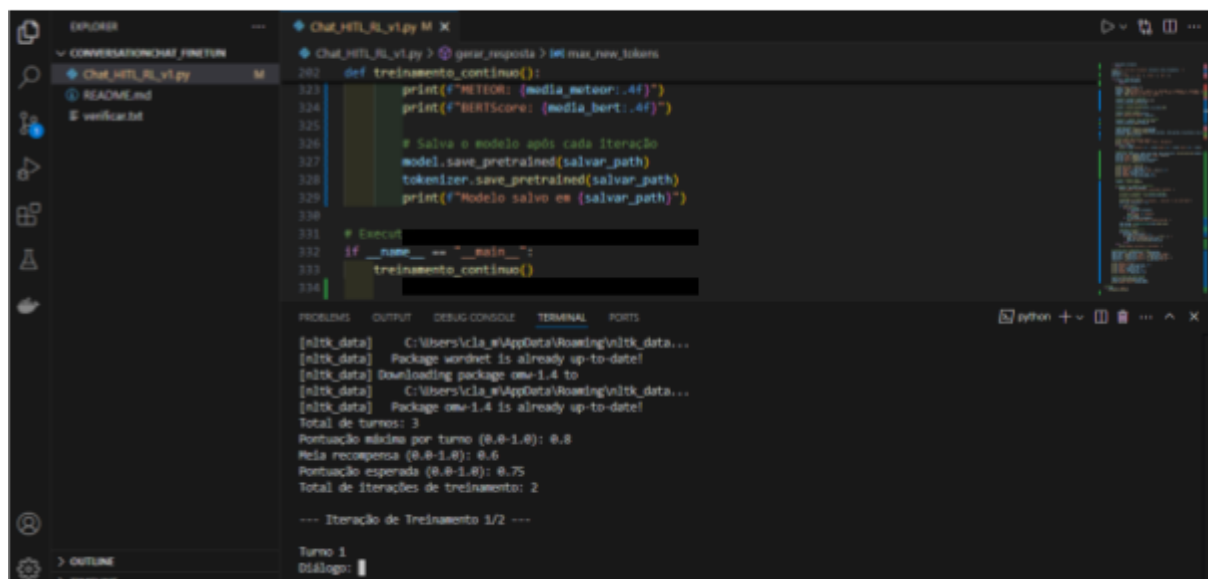
“Pontuação máxima por turno:” A pontuação que o professor e o aluno definem uma nota máxima desejada para que o modelo alcance para mostrar que está treinado e respondendo em ordem certa e com semântica adequada. Este é o princípio de ensino humano e de adestramento de animais, onde há uma recompensa ou não reforçando o ensino.

“Meia recompensa:” indica que quando qualquer palavra da frase de resposta do modelo estiver em posição diferente já que o ser humano costuma responder em diálogos com as palavras em posições iguais o modelo receberá menor pontuação. “O modelo DialoGPT, não utiliza sistema de recompensas, o objetivo é demonstrar o que há de básico em treinamento e caso utilize um outro modelo que use sistema de recompensas já estará definido no código”.

“Total de interação de treinamento” será o número de vezes que o modelo vai realizar o grupo de turnos de perguntas.

Obs: “1 turno” é igual a 2 diálogos, onde o Professor ou Aluno começam os diálogos, e o próximo turno o modelo responde, finaliza com quadro de notas de feedback humano e até terminar as iterações/repetições.

Fig.2



```
Chat_HTL_RL_v1.py M X
202 def treinamento_continuo():
223     print(f"METEOR: {media_meteor:.4f}")
224     print(f"BERTScore: {media_bert:.4f}")
225
226     # Salva o modelo após cada iteração
227     model.save_pretrained(salvar_path)
228     tokenizer.save_pretrained(salvar_path)
229     print(f"Modelo salvo em {salvar_path}")
230
231 # Execut
232 if __name__ == "__main__":
233     treinamento_continuo()
234
```

```
[nltk_data] C:\Users\clia\AppData\Local\Temp\nltk_data...
[nltk_data] Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data] C:\Users\clia\AppData\Local\Temp\nltk_data...
[nltk_data] Package omw-1.4 is already up-to-date!
Total de turnos: 3
Pontuação máxima por turno (0.0-1.0): 0.8
Meia recompensa (0.0-1.0): 0.6
Pontuação esperada (0.0-1.0): 0.75
Total de iterações de treinamento: 2

--- Iteração de Treinamento 1/2 ---

Turno 1
Diálogo: |
```

Fig.3

```

202 def treinamento_continuo():
203     print(f"Turno (turno + 1)")
204     dialogo = input("Diálogo: ")
205     intencao = input("Intenção (1 - Sugerir | 2 - Discordar | 3 - Perguntar | 4 - Concordar): ")
206     emocao = input("Emoção (0 - Neutro | 1 - Feliz | 2 - Triste | 3 - Surpreso | 4 - Animado | 5 - Raiva | 6 - Medo): ")
207     idioma = input("Idioma (en, pt, en-pt, pt-en): ")
208
209     # Adiciona ao contexto acumulado
210     contexto += dialogo + tokenizer.eos_token
211     contexto_acumulado.append(dialogo)
212
213     # Truncar o contexto se necessário
214     contexto = truncar_contexto(contexto, max_tokens=1000)
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

[nltk_data] C:\Users\cila\AppData\Local\nltk_data...
[nltk_data] Package cmu-1.4 is already up-to-date!
Total de turnos: 3
Pontuação máxima por turno (0.0-1.0): 0.8
Nota recompensa (0.0-1.0): 0.6
Pontuação esperada (0.0-1.0): 0.75
Total de iterações de treinamento: 2
--- Iteração de Treinamento 1/2 ---
Turno 1
Diálogo: Hey man , you wanna buy some weed ?
Intenção (1 - Sugerir | 2 - Discordar | 3 - Perguntar | 4 - Concordar): 3
Emoção (0 - Neutro | 1 - Feliz | 2 - Triste | 3 - Surpreso | 4 - Animado | 5 - Raiva | 6 - Medo): 0
Idioma (en, pt, en-pt, pt-en): en

```

Na figura 3 após o professor ou aluno inserir a frase do turno, é possível ver a informação de qual a iteração, e solicitando aos mesmos as configurações do turno para caso o modelo usado utilize para diálogos ações e emoções de como ele irá processar o idioma.

Nesta versão do DialoGPT utilizei estas configurações, há versões dele que não utilizam.

Intenção/ação o modelo vai entender o padrão para a ação a partir de informação numérica que o professor ou aluno indicam:

1 - Sugerir, 2 - Discordar, 3 - Perguntar, 4 - Concordar

Emoção/Sentimento o modelo vai entender o padrão para uma profundidade de como é expressado para um humano utilizando o mesmo princípio anterior para intenção/ação.

0 - Neutro, 1 - Feliz, 2 - Triste, 3 - Surpreso, 4 - Animado, 5 - Raiva, 6 - Medo

Idioma é usado para caso queira ensinar o modelo a ser um tradutor de inglês-português, ou vice-versa, ou se quer focar em diálogos em inglês ou português.

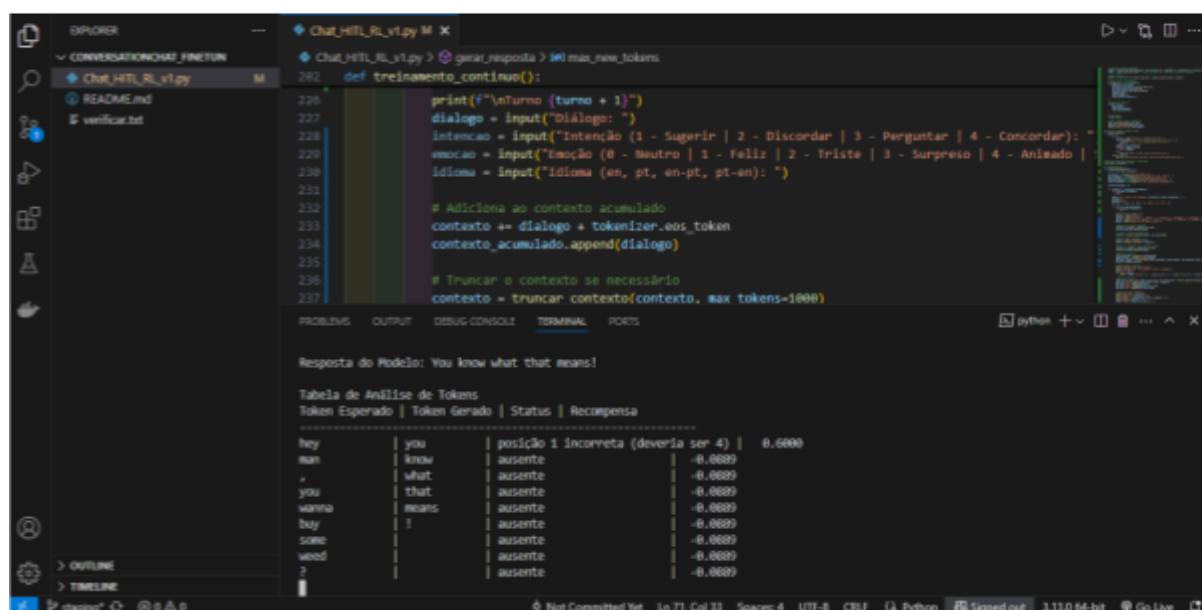
Adiante a Fig.3 mostra a tabela de comparação da frase gerada pelo professor e pelo modelo, indicando a semelhança na ordem dos tokens (palavras) com a frase humana.

Não é uma regra mas para treinar o modelo para aprender exatamente a sintaxe do idioma é importante que ele responda parecido o máximo de palavras e que tenha uma semântica ao mesmo tempo mas que não copie pois neste caso gera overfitting.

O que é Overfitting?

É quando um modelo de aprendizado de máquina aprende demais os dados de treinamento, incluindo ruídos e detalhes irrelevantes, em vez de capturar os padrões gerais. Como resultado, o modelo tem performance excelente nos dados de treino, mas péssima em dados novos (ex.: validação/teste) e ele "decora" os exemplos em vez de aprender a generalizar.

Fig.3



```
def treinamento_continuo():
    print(f" turno (turno + 1)")
    dialogo = input("Diálogo: ")
    intencao = input("Intenção (1 - Sugerir | 2 - Discordar | 3 - Perguntar | 4 - Concordar): ")
    emocao = input("Emoção (0 - Neutro | 1 - Feliz | 2 - Triste | 3 - Surpreso | 4 - Animado | ")
    idioma = input("Idioma (en, pt, en-pt, pt-en): ")

    # Adiciona ao contexto acumulado
    contexto += dialogo + tokenizer.eos_token
    contexto_acumulado.append(dialogo)

    # Truncar o contexto se necessário
    contexto = truncar_contexto(contexto, max_tokens=1000)
```

Resposta do Modelo: You know what that means!

Tabela de Análise de Tokens

Token Esperado	Token Gerado	Status	Recompensa
hey	you	posição 1 incorreta (deveria ser 4)	0.0000
man	know	ausente	-0.0009
,	what	ausente	-0.0009
you	that	ausente	-0.0009
wanna	means	ausente	-0.0009
buy	!	ausente	-0.0009
some		ausente	-0.0009
wend		ausente	-0.0009
?		ausente	-0.0009

O overfitting poderá ocorrer, já que não há variedade e densidade de dados a longo prazo, pois este método não entrega variedade de dados.

Este método deve ser usado somente para mostrar, em dinâmicas, buscando mais imersão já que o método mais adequado não vemos exatamente as coisas ocorrendo passo a passo, e a compreensão dos conceitos e ensino como um objeto de experimentação do treino de máquina deste modo é mais didático quando se pretende ensinar e fixar o conteúdo.

O modelo que menos apresentou este problema foi o Bert, Llama, e obteve excelentes resultados depois de meses diariamente conversando com o modelo onde ele aprendeu o português e inglês, com pouco overfitting.

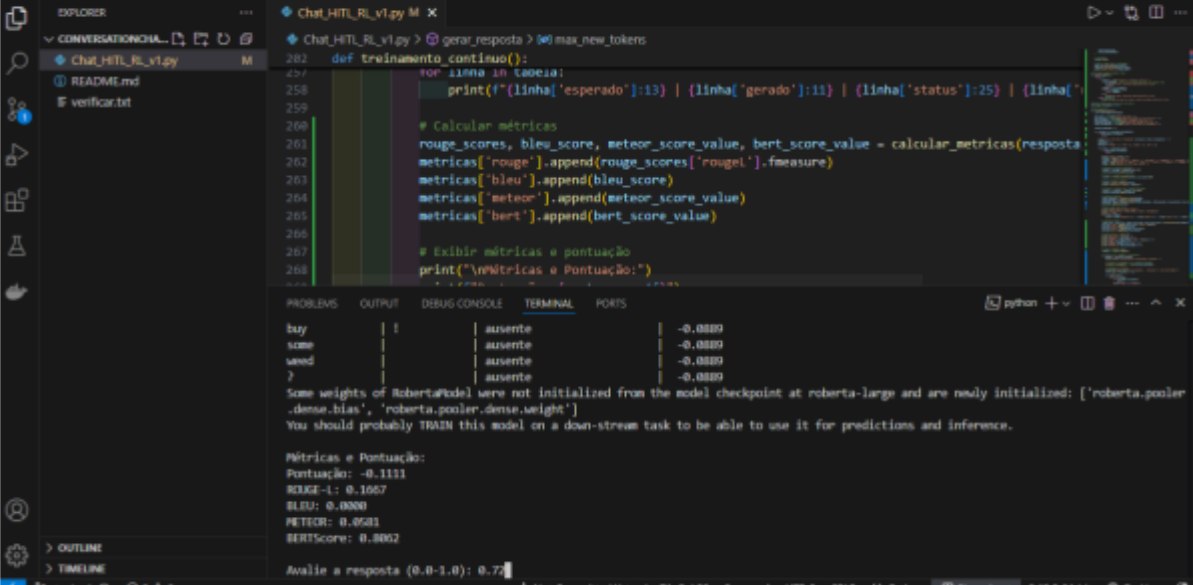
Na FIG.4 há informações automáticas de libraries (bibliotecas) de análise de modelos de conversação.

Pontuação: o indica a ordem da frase que o modelo gerou com a frase humana aplicada para dar o gatilho do diálogo, o ideal é ficar entre -25 e 25, muito alto indica imitação sem semântica alguma, muito baixo indica que o modelo não imita mas também sem semântica alguma.

ROUGE-L: é uma sigla para "Recall-Oriented Understudy for Gisting Evaluation - Longest Common Subsequence". Traduzido seria como uma avaliação de resumo baseada em Revocação com maior subsequência comum. Ele é um passo adiante na compreensão da ordem onde a tabela de pontuação explica a base do método. O ROUGE vai além: ele avalia a qualidade de um texto gerado (ex.: resumo, resposta) comparando-o com uma referência (ground truth), que no caso é a frase humana no turno 1 escrita pelo professor ou aluno, analisando a ordem e ocorrência dos tokens, sem interpretar seu significado linguístico, apenas com base em padrões estatísticos, pois as pessoas costumam não conversar com frases muito diferentes em sequência quando o assunto é focado. Ninguém conversa como o Mestre Yoda. Ele indicou 0,16 não está bom mas melhor que 0, é considerado satisfatório, o ideal seria próximo de 1.

BLEU: é uma sigla para "Bilingual Evaluation Understudy". Traduzido seria como um Avaliador Bilíngue de Substituto. Ele é um método clássico para medir a qualidade de traduções automáticas comparando-as com referências humanas. O BLEU vai direto ao ponto: avalia quantas palavras e sequências de palavras (n-grams) do texto gerado aparecem exatamente como nas traduções de referência, dando mais peso às correspondências exatas. Quanto mais parecido com a referência humana, maior a pontuação, que vai de 0 (péssimo) a 1 (perfeito). Ele não entende significado, apenas conta matches - por isso uma frase pode ter BLEU baixo mesmo sendo uma boa paráfrase. É como um corretor ortográfico que só aceita cópias fieis do que já viu antes. Nenhuma tradução real atinge 1, nem as humanas, porque sempre há variações possíveis. Da mesma forma a análise 0 para o exemplo não indicou nenhum problema logo não foi ruim pois a frase gerada pelo modelo fez sentido, mesmo diferente da referência humana.

Fig.4



The screenshot shows a Jupyter Notebook environment with a file explorer on the left, a code editor in the center, and a terminal at the bottom. The code in the editor is for a training loop and metric calculation. The terminal output shows the results of the training and evaluation metrics.

```
def treinamento_continuo():  
    for linha in dados:  
        print(f'({linha["esperado"]}:{linha["gerado"]}:{linha["status"]}:{linha["...  
        # Calcular métricas  
        rouge_scores, bleu_score, meteor_score_value, bert_score_value = calcular_metricas(resposta  
        metricas["rouge"].append(rouge_scores["rouge1"].fmeasure)  
        metricas["bleu"].append(bleu_score)  
        metricas["meteor"].append(meteor_score_value)  
        metricas["bert"].append(bert_score_value)  
        # Exibir métricas e pontuação  
        print("\nMétricas e Pontuação:")
```

Terminal Output:

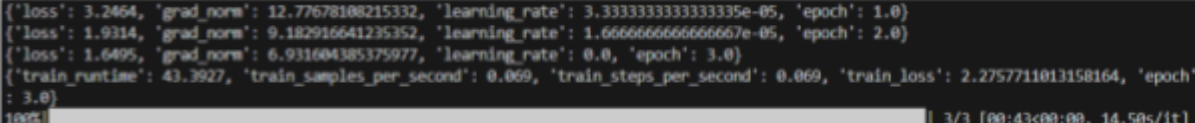
```
buy | | ausente | -0.0000  
some | | ausente | -0.0000  
wend | | ausente | -0.0000  
) | | ausente | -0.0000  
  
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['roberta.pooler  
.dense.bias', 'roberta.pooler.dense.weight']  
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.  
  
Métricas e Pontuação:  
Pontuação: -0.1111  
ROUGE-L: 0.1067  
BLEU: 0.0000  
METEOR: 0.0511  
BERTScore: 0.0002  
  
Avalie a resposta (0.0-1.0): 0.72
```

METEOR: é uma sigla para "Metric for Evaluation of Translation with Explicit ORdering". Traduzido seria como Métrica para Avaliação de Tradução com Ordenação Explícita. Ele é um passo além do BLEU, combinando precisão e recall com flexibilidade. O METEOR não só busca palavras idênticas às da referência, mas também aceita sinônimos e formas flexionadas como correspondências válidas - usando até dicionários e analisadores para isso. Além disso, ele penaliza menos a ordem diferente das palavras quando o sentido é preservado. A pontuação vai de 0 a 1, mas diferente do BLEU, aqui 1 é realmente alcançável quando a tradução é perfeita. Ele tenta capturar não só a forma, mas um pouco da essência do texto, ainda que de forma limitada. É como um corretor que aceita diferentes jeitos de dizer a mesma coisa, desde que faça sentido.

BERTScore: é uma métrica moderna que usa modelos de linguagem como o BERT para avaliar textos. Diferente do BLEU ou ROUGE que contam palavras exatas, o BERTScore compara o significado das frases. Ele pega cada palavra do texto gerado e da referência, transforma em vetores (representações numéricas do significado) e calcula quão similares são esses vetores - capturando sinônimos, paráfrases e até relações contextuais que outras métricas perdem. O resultado é um score entre 0 e 1, onde valores mais altos indicam maior similaridade semântica. A grande vantagem? Ele entende que "cachorro" e "animal de estimação" podem ser equivalentes dependendo do contexto, coisa que métricas tradicionais nunca aceitariam. Mas tem um porém: como usa modelos complexos, exige mais poder computacional que BLEU/ROUGE. É como um professor que lê suas redações pensando no que você quis dizer, não só em quantas palavras da resposta modelo você acertou.

Na FIG.5 após o modelo não estar satisfatório, recebendo nota de 0.72, e o professor no início tinha definido 0.8 como mínimo para ir para próxima iteração, e segundo o feedback humano depois de avaliar as métricas, mesmo o BERTscore estando bom automaticamente o código aplica treinamento no modelo. As métricas são mais apropriadas para análise de quem está treinando o modelo.

Fig.5



```
{ 'loss': 3.2464, 'grad_norm': 12.77678108215332, 'learning_rate': 3.3333333333333335e-05, 'epoch': 1.0}
{'loss': 1.9314, 'grad_norm': 9.182916641235352, 'learning_rate': 1.6666666666666667e-05, 'epoch': 2.0}
{'loss': 1.6495, 'grad_norm': 6.931604385375977, 'learning_rate': 0.0, 'epoch': 3.0}
{'train_runtime': 43.3927, 'train_samples_per_second': 0.069, 'train_steps_per_second': 0.069, 'train_loss': 2.2757711013158164, 'epoch': 3.0}
100% | 3/3 [00:43<00:00, 14.50s/it]
```

Alguns termos são indicados:

Epoch: Quantas vezes o modelo viu todo o conjunto de dados de treino. Como reler um livro 3 vezes para entender melhor.

**Loss:** Representa o valor da função de perda, e está em 3.24, é alto no início mas o importante é o quanto chegou no último epoch, ou seja, indica quão "erradas" estão as previsões do modelo em relação aos rótulos reais ou às saídas do modelo professor (no caso da destilação), então se caiu para 1.64, e isto é bom.

**Grad\_norm:** Norma (magnitude) dos gradientes durante a atualização dos parâmetros do modelo, ou seja, indicam a direção e intensidade dos ajustes nos pesos do modelo para minimizar a loss. Maior que 100 o gradiente é instável, e menor que 0 indica que não aprendeu nada, está em 12 no primeiro epoch e depois caiu para 6, está regular, deveria ter ficado fixado, se ocorrer mais epochs pode ficar crítico, pois tende a cair sempre, mas não indicam sem explosão/vanishing.

**Learning\_rate:** Taxa de aprendizagem (learning rate) usada pelo otimizador, ou seja, controla o tamanho dos passos que o modelo dá para ajustar seus parâmetros durante o treinamento. Vemos o valor de  $3.33 \cdot 10^{-5}$  (ou 0.0000333) é relativamente baixo, típico em etapas de ajuste fino (fine-tuning) ou treinamento estável. Indica que vai demorar treinar e muito.

Após pode ver um resumo:

**Train\_runtime (43.39s):** Tempo total que o treinamento levou para ser concluído. Como um cronômetro marcando a duração total, na verdade é maior, pois este cálculo é feito pelo modelo desconsiderando processamento dos dados, pre-processamento de tokens e embeddings e outros, e somente a entrada e a saída. Somando as 3 epochs chega a uns 2,17 por epoch o que seria 7 minutos em média.

**Train\_samples\_per\_second (0.069):** Quantas amostras de dados o modelo consegue processar por segundo. Parecido com quantas páginas um escritor consegue revisar por minuto - quanto maior, mais rápido.

**Train\_steps\_per\_second (0.069):** Quantos passos de atualização do modelo são feitos por segundo. Como um marcador de passos - cada passo é uma pequena melhoria no modelo.

**Train\_loss (2.275):** Mede o erro do modelo durante o treino. Quanto menor, melhor o modelo está aprendendo. Como a pontuação de erros em um teste - você quer diminuir.

O código:

```
import json
from transformers import AutoTokenizer, AutoModelForCausalLM, Trainer,
TrainingArguments
import torch
from torch.utils.data import Dataset
from rouge_score import rouge_scorer
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction
from nltk.translate.meteor_score import meteor_score
from bert_score import score
import argparse
import nltk
import re
import time
import threading

# Baixar recursos do NLTK
nltk.download('wordnet')
nltk.download('omw-1.4')

# Configuração de hiperparâmetros
parser = argparse.ArgumentParser()
parser.add_argument("--batch_size", type=int, default=1)
parser.add_argument("--learning_rate", type=float, default=5e-5)
parser.add_argument("--num_train_epochs", type=int, default=3)
parser.add_argument("--max_length", type=int, default=50)
parser.add_argument("--min_length", type=int, default=1)
parser.add_argument("--do_sample", type=bool, default=True)
parser.add_argument("--temperature", type=float, default=0.7)
parser.add_argument("--top_k", type=int, default=50)
parser.add_argument("--top_p", type=float, default=0.9)
parser.add_argument("--repetition_penalty", type=float, default=1.2)
parser.add_argument("--num_return_sequences", type=int, default=1)
parser.add_argument("--pad_token_id", type=int, default=50256)
parser.add_argument("--eos_token_id", type=int, default=50256)

args = parser.parse_args()

# Caminhos para salvar e carregar os modelos
modelo_path = r"copie e cole o endereço da pasta que foi alocado o modelo para leitura"
salvar_path = r"copie e cole o endereço da pasta que foi alocado o modelo para salvar"

# Carregar modelo e tokenizer
tokenizer = AutoTokenizer.from_pretrained(modelo_path)
model = AutoModelForCausalLM.from_pretrained(modelo_path)
```



```

# Configurar o token de padding, se necessário
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token # Usa o token de fim de sequência como padding

# Mover o modelo para o dispositivo (CPU ou GPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# Classe para o dataset de conversa
class ConversaDataset(Dataset):
    def __init__(self, inputs, masks, labels):
        self.inputs = inputs
        self.masks = masks
        self.labels = labels

    def __len__(self):
        return len(self.inputs)

    def __getitem__(self, idx):
        return {
            "input_ids": self.inputs[idx],
            "attention_mask": self.masks[idx],
            "labels": self.labels[idx]
        }

# Função para gerar resposta contextualizada
def gerar_resposta(contexto, max_new_tokens=100, temperature=0.7, top_k=50, top_p=0.9):
    input_ids = tokenizer.encode(contexto + tokenizer.eos_token,
    return_tensors="pt").to(device)

    # Verifica se o tokenizer tem um token de padding
    pad_token_id = tokenizer.pad_token_id if tokenizer.pad_token_id is not None else
    tokenizer.eos_token_id

    resposta_ids = model.generate(
        input_ids,
        max_new_tokens=max_new_tokens,
        pad_token_id=pad_token_id, # Usa o token de padding ou EOS
        do_sample=True,
        temperature=temperature,
        top_k=top_k,
        top_p=top_p,
        repetition_penalty=args.repetition_penalty,
        attention_mask=input_ids.ne(pad_token_id).float().to(device) # Usa o token de
padding ou EOS

```

```

)
        resposta = tokenizer.decode(resposta_ids[:, input_ids.shape[-1]:][0],
skip_special_tokens=True)
    return resposta

```

# Função de tokenização detalhada

```

def tokenizar_frase(frase):
    tokens = re.findall(r"\w+|[\^\w\s]", frase.lower())
    return tokens

```

# Função de análise comparativa

```

def analisar_tokens(tokens_esperados, tokens_gerados, max_pontuacao,
meia_recompensa):
    recompensa_por_token = max_pontuacao / len(tokens_esperados) if tokens_esperados
else 0
    penalidade = -recompensa_por_token

```

```

tabela = []
pontuacao_total = 0

```

# Preencher tokens ausentes com strings vazias

```

max_tokens = max(len(tokens_esperados), len(tokens_gerados))
tokens_esperados += [""] * (max_tokens - len(tokens_esperados))
tokens_gerados += [""] * (max_tokens - len(tokens_gerados))

```

```

for i, (te, tg) in enumerate(zip(tokens_esperados, tokens_gerados)):

```

```

    if tg == te:

```

```

        status = f"posição {i+1} correta"

```

```

        recompensa = recompensa_por_token

```

```

    elif tg in tokens_esperados:

```

```

        pos_correta = tokens_esperados.index(tg)

```

```

        status = f"posição {i+1} incorreta (deveria ser {pos_correta+1})"

```

```

        recompensa = meia_recompensa

```

```

    else:

```

```

        status = "ausente"

```

```

        recompensa = penalidade

```

```

    pontuacao_total += recompensa

```

```

    tabela.append({

```

```

        'esperado': te,

```

```

        'gerado': tg,

```

```

        'status': status,

```

```

        'recompensa': round(recompensa, 4)

```

```

    })

```

```

return pontuacao_total, tabela

```

# Função para calcular métricas

```

def calcular_metricas(resposta_gerada, resposta_esperada):
    scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'], use_stemmer=True)
    rouge_scores = scorer.score(resposta_esperada, resposta_gerada)
    smoothing_function = SmoothingFunction().method1
    bleu_score = sentence_bleu([resposta_esperada.split()], resposta_gerada.split(),
    smoothing_function=smoothing_function)
    meteor_score_value = meteor_score([resposta_esperada.split()], resposta_gerada.split())
    bert_score_value = score([resposta_gerada], [resposta_esperada],
    lang="en")[2].mean().item()
    return rouge_scores, bleu_score, meteor_score_value, bert_score_value

def trincar_contexto(contexto, max_tokens=1000):
    tokens = tokenizer.encode(contexto)
    if len(tokens) > max_tokens:
        tokens = tokens[-max_tokens:] # Mantém os últimos `max_tokens` tokens
    return tokenizer.decode(tokens)

# Função para processar o treinamento
def processar_treinamento(contexto_acumulado):
    # Tokeniza o contexto acumulado
    inputs = tokenizer(contexto_acumulado, return_tensors="pt", padding=True,
    truncation=True).to(device)
    labels = inputs.input_ids.clone() # Labels são iguais aos inputs para modelos de
    linguagem

    # Cria o dataset
    dataset = ConversaDataset([inputs.input_ids], [inputs.attention_mask], [labels])

    # Configurações de treinamento
    training_args = TrainingArguments(
        output_dir=salvar_path,
        per_device_train_batch_size=args.batch_size,
        learning_rate=args.learning_rate,
        num_train_epochs=args.num_train_epochs,
        logging_steps=1,
        save_strategy="no",
        remove_unused_columns=False
    )

    # Inicializa o Trainer
    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=dataset
    )

    # Treinamento
    trainer.train()

```

```

# Salva o modelo após o treinamento
model.save_pretrained(salvar_path)
tokenizer.save_pretrained(salvar_path)
print(f"Modelo salvo em {salvar_path}")

# Função para coletar feedback humano
def coletar_feedback():
    while True:
        try:
            feedback = input("\nAvalie a resposta (0.0-1.0): ")
            if not feedback: # Se o usuário pressionar Enter sem digitar nada
                print("Por favor, insira um valor entre 0.0 e 1.0.")
                continue
            feedback = float(feedback)
            if 0.0 <= feedback <= 1.0:
                return feedback
            else:
                print("Por favor, insira um valor entre 0.0 e 1.0.")
        except ValueError:
            print("Entrada inválida. Por favor, insira um número entre 0.0 e 1.0.")

# Variável global para controle de interrupção
interromper_treinamento = False

# Função principal
def treinamento_continuo():
    global interromper_treinamento

    num_turnos = int(input("Total de turnos: "))
    max_pontuacao = float(input("Pontuação máxima por turno (0.0-1.0): "))
    meia_recompensa = float(input("Meia recompensa (0.0-1.0): "))
    pontuacao_esperada = float(input("Pontuação esperada (0.0-1.0): "))
    total_treinamentos = int(input("Total de iterações de treinamento: "))

    contexto_acumulado = []

    for iteracao in range(total_treinamentos):
        if interromper_treinamento:
            break

        print(f"\n--- Iteração de Treinamento {iteracao+1}/{total_treinamentos} ---")
        contexto = ""
        pontuacoes = []
        metricas = {'rouge': [], 'bleu': [], 'meteor': [], 'bert': []}

        for turno in range(num_turnos):
            if interromper_treinamento:

```

```

        break

    print(f"\nTurno {turno + 1}")
    dialogo = input("Diálogo: ")
    intencao = input("Intenção (1 - Sugerir | 2 - Discordar | 3 - Perguntar | 4 - Concordar): ")

    emocao = input("Emoção (0 - Neutro | 1 - Feliz | 2 - Triste | 3 - Surpreso | 4 - Animado | 5 - Raiva | 6 - Medo): ")
    idioma = input("Idioma (en, pt, en-pt, pt-en): ")

    # Adiciona ao contexto acumulado
    contexto += dialogo + tokenizer.eos_token
    contexto_acumulado.append(dialogo)

    # Truncar o contexto se necessário
    contexto = truncar_contexto(contexto, max_tokens=1000)

    # Gerar resposta do modelo
    resposta = gerar_resposta(contexto)
    print(f"\nResposta do Modelo: {resposta}")

    # Atualiza o contexto com a resposta do modelo
    contexto += resposta + tokenizer.eos_token
    contexto_acumulado.append(resposta)

    # Tokeniza e analisa
    tokens_esperados = tokenizar_frase(dialogo)
    tokens_gerados = tokenizar_frase(resposta)
    pontuacao, tabela = analisar_tokens(tokens_esperados, tokens_gerados,
max_pontuacao, meia_recompensa)
    pontuacoes.append(pontuacao)

    # Mostra tabela de análise
    print("\nTabela de Análise de Tokens")
    print("Token Esperado | Token Gerado | Status | Recompensa")
    print("-" * 60)
    for linha in tabela:
        print(f"{linha['esperado']:13} | {linha['gerado']:11} | {linha['status']:25} | {linha['recompensa']:8.4f}")

    # Calcular métricas
    rouge_scores, bleu_score, meteor_score_value, bert_score_value =
calcular_metricas(resposta, dialogo)
    metricas['rouge'].append(rouge_scores['rougeL'].fmeasure)
    metricas['bleu'].append(bleu_score)
    metricas['meteor'].append(meteor_score_value)
    metricas['bert'].append(bert_score_value)

```

```

# Exibir métricas e pontuação
print("\nMétricas e Pontuação:")
print(f"Pontuação: {pontuacao:.4f}")
print(f"ROUGE-L: {rouge_scores['rougeL'].fmeasure:.4f}")
print(f"BLEU: {bleu_score:.4f}")
print(f"METEOR: {meteor_score_value:.4f}")
print(f"BERTScore: {bert_score_value:.4f}")

# Coletar feedback humano
feedback = coletar_feedback()

# Verificar se o feedback é satisfatório
if feedback < pontuacao_esperada:
    print("Feedback insatisfatório. Reiniciando o processo...")

# Processa o treinamento com o contexto acumulado
processar_treinamento(" ".join(contexto_acumulado))

# Contagem de 5 segundos
print("\nPróxima iteração em 5 segundos... (pressione 's' para interromper)")
interromper_treinamento = False

def temporizador():
    for i in range(6):
        if interromper_treinamento:
            break
        print(f"Tempo: {i} segundos")
        time.sleep(1)
    if not interromper_treinamento:
        print("Continuando para a próxima iteração...")

timer_thread = threading.Thread(target=temporizador)
timer_thread.start()

# Aguarda input do usuário
user_input = input()
if user_input.lower() == 's':
    interromper_treinamento = True
    if input("Salvar progresso? (s/n): ").lower() == 's':
        model.save_pretrained(salvar_path)
        tokenizer.save_pretrained(salvar_path)
    return
else:
    print("Feedback satisfatório. Continuando...")

# Calcular média de pontuação e métricas
media_pontuacao = sum(pontuacoes) / len(pontuacoes)
media_rouge = sum(metricas['rouge']) / len(metricas['rouge'])

```

```
media_bleu = sum(metricas['bleu']) / len(metricas['bleu'])
media_meteor = sum(metricas['meteor']) / len(metricas['meteor'])
media_bert = sum(metricas['bert']) / len(metricas['bert'])

print("\nMédias Finais:")
print(f"Pontuação: {media_pontuacao:.4f}")
print(f"ROUGE-L: {media_rouge:.4f}")
print(f"BLEU: {media_bleu:.4f}")
print(f"METEOR: {media_meteor:.4f}")
print(f"BERTScore: {media_bert:.4f}")

# Salva o modelo após cada iteração
model.save_pretrained(salvar_path)
tokenizer.save_pretrained(salvar_path)
print(f"Modelo salvo em {salvar_path}")

# Executar
if __name__ == "__main__":
    treinamento_continuo()
```

Base Teórica:

DialoGPT

Zhang, Y. et al. (2020). "DialoGPT: Large-Scale Generative Pre-training for Conversational Response Generation".

arXiv preprint: arXiv:1911.00536

Disponível em: [arxiv.org/abs/1911.00536](https://arxiv.org/abs/1911.00536)

Aprendizado por Reforço para Diálogo

Li, J. et al. (2016). "Deep Reinforcement Learning for Dialogue Generation".

Proceedings of EMNLP 2016, pp. 1192-1202.

DOI: 10.18653/v1/D16-1127

Avaliação de Diálogos

Liu, C. et al. (2016). "How NOT To Evaluate Your Dialogue System".

Proceedings of EMNLP 2016, pp. 2122-2132.

DOI: 10.18653/v1/D16-1230

Técnicas Complementares

Decoding Strategies

Holtzman, A. et al. (2020). "The Curious Case of Neural Text Degeneration".

ICLR 2020.

arXiv preprint: arXiv:1904.09751

Transfer Learning para NLP

Devlin, J. et al. (2019). "BERT: Pre-training of Deep Bidirectional Transformers".

\*NAACL-HLT 2019\*.

DOI: 10.18653/v1/N19-1423

Métricas de Avaliação

Papineni, K. et al. (2002). "BLEU: a Method for Automatic Evaluation of Machine Translation".

ACL 2002, pp. 311-318.

DOI: 10.3115/1073083.1073135



