

Cheat Sheet: Generative AI Advanced Fine-Tuning for LLMs

Package/Method	Description	Code Example
CUDA-compatible GPU	<p>Available in the system using PyTorch, a popular deep-learning framework. If a GPU is available, it assigns the device variable to "cuda" (CUDA, the parallel computing platform and application programming interface model developed by NVIDIA). If a GPU is not available, it assigns the device variable to "cpu" (which means the code will run on the CPU instead).</p>	<pre> 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 11. 11 12. 12 13. 13 14. 14 15. 15 16. 16 17. 17 18. 18 19. 19 20. 20 21. 21 22. 22 23. 23 24. 24 25. 25 26. 26 27. 27 28. 28 29. 29 30. 30 31. 31 32. 32 33. 33 34. 34 35. 35 36. 36 37. 37 38. 38 39. 39 40. 40 41. 41 42. 42 43. 43 44. 44 45. 45 46. 46 47. 47 48. 48 49. 49 50. 50 51. 51 52. 52 53. 53 54. 54 55. 55 56. 56 57. 57 58. 58 59. 59 60. 60 61. 61 62. 62 63. 63 64. 64 65. 65 66. 66 67. 67 68. 68 69. 69 70. 70 71. 71 72. 72 73. 73 74. 74 75. 75 76. 76 77. 77 78. 78 79. 79 80. 80 81. 81 82. 82 83. 83 84. 84 85. 85 86. 86 87. 87 88. 88 89. 89 90. 90 91. 91 92. 92 93. 93 94. 94 95. 95 96. 96 97. 97 98. 98 99. 99 100. 100 101. 101 102. 102 103. 103 104. 104 105. 105 106. 106 107. 107 108. 108 109. 109 110. 110 111. 111 112. 112 113. 113 114. 114 115. 115 116. 116 117. 117 118. 118 119. 119 120. 120 121. 121 122. 122 123. 123 124. 124 125. 125 126. 126 127. 127 128. 128 129. 129 130. 130 131. 131 132. 132 133. 133 134. 134 135. 135 136. 136 137. 137 138. 138 139. 139 140. 140 141. 141 142. 142 143. 143 144. 144 145. 145 146. 146 147. 147 148. 148 149. 149 150. 150 151. 151 152. 152 153. 153 154. 154 155. 155 156. 156 157. 157 158. 158 159. 159 160. 160 161. 161 162. 162 163. 163 164. 164 165. 165 166. 166 167. 167 168. 168 169. 169 170. 170 171. 171 172. 172 173. 173 174. 174 175. 175 176. 176 177. 177 178. 178 179. 179 180. 180 181. 181 182. 182 183. 183 184. 184 185. 185 186. 186 187. 187 188. 188 189. 189 190. 190 191. 191 192. 192 193. 193 194. 194 195. 195 196. 196 197. 197 198. 198 199. 199 200. 200 201. 201 202. 202 203. 203 204. 204 205. 205 206. 206 207. 207 208. 208 209. 209 210. 210 211. 211 212. 212 213. 213 214. 214 215. 215 216. 216 217. 217 218. 218 219. 219 220. 220 221. 221 222. 222 223. 223 224. 224 225. 225 226. 226 227. 227 228. 228 229. 229 230. 230 231. 231 232. 232 233. 233 234. 234 235. 235 236. 236 237. 237 238. 238 239. 239 240. 240 241. 241 242. 242 243. 243 244. 244 245. 245 246. 246 247. 247 248. 248 249. 249 250. 250 251. 251 252. 252 253. 253 254. 254 255. 255 256. 256 257. 257 258. 258 259. 259 260. 260 261. 261 262. 262 263. 263 264. 264 265. 265 266. 266 267. 267 268. 268 269. 269 270. 270 271. 271 272. 272 273. 273 274. 274 275. 275 276. 276 277. 277 278. 278 279. 279 280. 280 281. 281 282. 282 283. 283 284. 284 285. 285 286. 286 287. 287 288. 288 289. 289 290. 290 291. 291 292. 292 293. 293 294. 294 295. 295 296. 296 297. 297 298. 298 299. 299 300. 300 301. 301 302. 302 303. 303 304. 304 305. 305 306. 306 307. 307 308. 308 309. 309 310. 310 311. 311 312. 312 313. 313 314. 314 315. 315 316. 316 317. 317 318. 318 319. 319 320. 320 321. 321 322. 322 323. 323 324. 324 325. 325 326. 326 327. 327 328. 328 329. 329 330. 330 331. 331 332. 332 333. 333 334. 334 335. 335 336. 336 337. 337 338. 338 339. 339 340. 340 341. 341 342. 342 343. 343 344. 344 345. 345 346. 346 347. 347 348. 348 349. 349 350. 350 351. 351 352. 352 353. 353 354. 354 355. 355 356. 356 357. 357 358. 358 359. 359 360. 360 361. 361 362. 362 363. 363 364. 364 365. 365 366. 366 367. 367 368. 368 369. 369 370. 370 371. 371 372. 372 373. 373 374. 374 375. 375 376. 376 377. 377 378. 378 379. 379 380. 380 381. 381 382. 382 383. 383 384. 384 385. 385 386. 386 387. 387 388. 388 389. 389 390. 390 391. 391 392. 392 393. 393 394. 394 395. 395 396. 396 397. 397 398. 398 399. 399 400. 400 401. 401 402. 402 403. 403 404. 404 405. 405 406. 406 407. 407 408. 408 409. 409 410. 410 411. 411 412. 412 413. 413 414. 414 415. 415 416. 416 417. 417 418. 418 419. 419 420. 420 421. 421 422. 422 423. 423 424. 424 425. 425 426. 426 427. 427 428. 428 429. 429 430. 430 431. 431 432. 432 433. 433 434. 434 435. 435 436. 436 437. 437 438. 438 439. 439 440. 440 441. 441 442. 442 443. 443 444. 444 445. 445 446. 446 447. 447 448. 448 449. 449 450. 450 451. 451 452. 452 453. 453 454. 454 455. 455 456. 456 457. 457 458. 458 459. 459 460. 460 461. 461 462. 462 463. 463 464. 464 465. 465 466. 466 467. 467 468. 468 469. 469 470. 470 471. 471 472. 472 473. 473 474. 474 475. 475 476. 476 477. 477 478. 478 479. 479 480. 480 481. 481 482. 482 483. 483 484. 484 485. 485 486. 486 487. 487 488. 488 489. 489 490. 490 491. 491 492. 492 493. 493 494. 494 495. 495 496. 496 497. 497 498. 498 499. 499 500. 500 501. 501 502. 502 503. 503 504. 504 505. 505 506. 506 507. 507 508. 508 509. 509 510. 510 511. 511 512. 512 513. 513 514. 514 515. 515 516. 516 517. 517 518. 518 519. 519 520. 520 521. 521 522. 522 523. 523 524. 524 525. 525 526. 526 527. 527 528. 528 529. 529 530. 530 531. 531 532. 532 533. 533 534. 534 535. 535 536. 536 537. 537 538. 538 539. 539 540. 540 541. 541 542. 542 543. 543 544. 544 545. 545 546. 546 547. 547 548. 548 549. 549 550. 550 551. 551 552. 552 553. 553 554. 554 555. 555 556. 556 557. 557 558. 558 559. 559 560. 560 561. 561 562. 562 563. 563 564. 564 565. 565 566. 566 567. 567 568. 568 569. 569 570. 570 571. 571 572. 572 573. 573 574. 574 575. 575 576. 576 577. 577 578. 578 579. 579 580. 580 581. 581 582. 582 583. 583 584. 584 585. 585 586. 586 587. 587 588. 588 589. 589 590. 590 591. 591 592. 592 593. 593 594. 594 595. 595 596. 596 597. 597 598. 598 599. 599 600. 600 601. 601 602. 602 603. 603 604. 604 605. 605 606. 606 607. 607 608. 608 609. 609 610. 610 611. 611 612. 612 613. 613 614. 614 615. 615 616. 616 617. 617 618. 618 619. 619 620. 620 621. 621 622. 622 623. 623 624. 624 625. 625 626. 626 627. 627 628. 628 629. 629 630. 630 631. 631 632. 632 633. 633 634. 634 635. 635 636. 636 637. 637 638. 638 639. 639 640. 640 641. 641 642. 642 643. 643 644. 644 645. 645 646. 646 647. 647 648. 648 649. 649 650. 650 651. 651 652. 652 653. 653 654. 654 655. 655 656. 656 657. 657 658. 658 659. 659 660. 660 661. 661 662. 662 663. 663 664. 664 665. 665 666. 666 667. 667 668. 668 669. 669 670. 670 671. 671 672. 672 673. 673 674. 674 675. 675 676. 676 677. 677 678. 678 679. 679 680. 680 681. 681 682. 682 683. 683 684. 684 685. 685 686. 686 687. 687 688. 688 689. 689 690. 690 691. 691 692. 692 693. 693 694. 694 695. 695 696. 696 697. 697 698. 698 699. 699 700. 700 701. 701 702. 702 703. 703 704. 704 705. 705 706. 706 707. 707 708. 708 709. 709 710. 710 711. 711 712. 712 713. 713 714. 714 715. 715 716. 716 717. 717 718. 718 719. 719 720. 720 721. 721 722. 722 723. 723 724. 724 725. 725 726. 726 727. 727 728. 728 729. 729 730. 730 731. 731 732. 732 733. 733 734. 734 735. 735 736. 736 737. 737 738. 738 739. 739 740. 740 741. 741 742. 742 743. 743 744. 744 745. 745 746. 746 747. 747 748. 748 749. 749 750. 750 751. 751 752. 752 753. 753 754. 754 755. 755 756. 756 757. 757 758. 758 759. 759 760. 760 761. 761 762. 762 763. 763 764. 764 765. 765 766. 766 767. 767 768. 768 769. 769 770. 770 771. 771 772. 772 773. 773 774. 774 775. 775 776. 776 777. 777 778. 778 779. 779 780. 780 781. 781 782. 782 783. 783 784. 784 785. 785 786. 786 787. 787 788. 788 789. 789 790. 790 791. 791 792. 792 793. 793 794. 794 795. 795 796. 796 797. 797 798. 798 799. 799 800. 800 801. 801 802. 802 803. 803 804. 804 805. 805 806. 806 807. 807 808. 808 809. 809 810. 810 811. 811 812. 812 813. 813 814. 814 815. 815 816. 816 817. 817 818. 818 819. 819 820. 820 821. 821 822. 822 823. 823 824. 824 825. 825 826. 826 827. 827 828. 828 829. 829 830. 830 831. 831 832. 832 833. 833 834. 834 835. 835 836. 836 837. 837 838. 838 839. 839 840. 840 841. 841 842. 842 843. 843 844. 844 845. 845 846. 846 847. 847 848. 848 849. 849 850. 850 851. 851 852. 852 853. 853 854. 854 855. 855 856. 856 857. 857 858. 858 859. 859 860. 860 861. 861 862. 862 863. 863 864. 864 865. 865 866. 866 867. 867 868. 868 869. 869 870. 870 871. 871 872. 872 873. 873 874. 874 875. 875 876. 876 877. 877 878. 878 879. 879 880. 880 881. 881 882. 882 883. 883 884. 884 885. 885 886. 886 887. 887 888. 888 889. 889 890. 890 891. 891 892. 892 893. 893 894. 894 895. 895 896. 896 897. 897 898. 898 899. 899 900. 900 901. 901 902. 902 903. 903 904. 904 905. 905 906. 906 907. 907 908. 908 909. 909 910. 910 911. 911 912. 912 913. 913 914. 914 915. 915 916. 916 917. 917 918. 918 919. 919 920. 920 921. 921 922. 922 923. 923 924. 924 925. 925 926. 926 927. 927 928. 928 929. 929 930. 930 931. 931 932. 932 933. 933 934. 934 935. 935 936. 936 937. 937 938. 938 939. 939 940. 940 941. 941 942. 942 943. 943 944. 944 945. 945 946. 946 947. 947 948. 948 949. 949 950. 950 951. 951 952. 952 953. 953 954. 954 955. 955 956. 956 957. 957 958. 958 959. 959 960. 960 961. 961 962. 962 963. 963 964. 964 965. 965 966. 966 967. 967 968. 968 969. 969 970. 970 971. 971 972. 972 973. 973 974. 974 975. 975 976. 976 977. 977 978. 978 979. 979 980. 980 981. 981 982. 982 983. 983 984. 984 985. 985 986. 986 987. 987 988. 988 989. 989 990. 990 991. 991 992. 992 993. 993 994. 994 995. 995 996. 996 997. 997 998. 998 999. 999 1000. 1000 1001. 1001 1002. 1002 1003. 1003 1004. 1004 1005. 1005 1006. 1006 1007. 1007 1008. 1008 1009. 1009 1010. 1010 1011. 1011 1012. 1012 1013. 1013 1014. 1014 1015. 1015 1016. 1016 1017. 1017 1018. 1018 1019. 1019 1020. 1020 1021. 1021 1022. 1022 1023. 1023 1024. 1024 1025. 1025 1026. 1026 1027. 1027 1028. 1028 1029. 1029 1030. 1030 1031. 1031 1032. 1032 1033. 1033 1034. 1034 1035. 1035 1036. 1036 1037. 1037 1038. 1038 1039. 1039 1040. 1040 1041. 1041 1042. 1042 1043. 1043 1044. 1044 1045. 1045 1046. 1046 1047. 1047 1048. 1048 1049. 1049 1050. 1050 1051. 1051 1052. 1052 1053. 1053 1054. 1054 1055. 1055 1056. 1056 1057. 1057 1058. 1058 1059. 1059 1060. 1060 1061. 1061 1062. 1062 1063. 1063 1064. 1064 1065. 1065 1066. 1066 1067. 1067 1068. 1068 1069. 1069 1070. 1070 1071. 1071 1072. 1072 1073. 1073 1074. 1074 1075. 1075 1076. 1076 1077. 1077 1078. 1078 1079. 1079 1080. 1080 1081. 1081 1082. 1082 1083. 1083 1084. 1084 1085. 1085 1086. 1086 1087. 1087 1088. 1088 1089. 1089 1090. 1090 1091. 1091 1092. 1092 1093. 1093 1094. 1094 1095. 1095 1096. 1096 1097. 1097 1098. 1098 1099. 1099 1100. 1100 1101. 1101 1102. 1102 1103. 1103 1104. 1104 1105. 1105 1106. 1106 1107. 1107 1108. 1108 1109. 1109 1110. 1110 1111. 1111 1112. 1112 1113. 1113 1114. 1114 1115. 1115 1116. 1116 1117. 1117 1118. 1118 1119. 1119 1120. 1120 1121. 1121 1122. 1122 1123. 1123 1124. 1124 1125. 1125 1126. 1126 1127. 1127 1128. 1128 1129. 1129 1130. 1130 1131. 1131 1132. 1132 1133. 1133 1134. 1134 1135. 1135 1136. 1136 1137. 1137 1138. 1138 1139. 1139 1140. 1140 1141. 1141 1142. 1142 1143. 1143 1144. 1144 1145. 1145 1146. 1146 1147. 1147 1148. 1148 1149. 1149 1150. 1150 1151. 1151 1152. 1152 1153. 1153 1154. 1154 1155. 1155 1156. 1156 1157. 1157 1158. 1158 1159. 1159 1160. 1160 1161. 1161 1162. 1162 1163. 1163 1164. 1164 1165. 1165 1166. 1166 1167. 1167 1168. 1168 1169. 1169 1170. 1170 1171. 1171 1172. 1172 1173. 1173 1174. 1174 1175. 1175 1176. 1176 1177. 1177 1178. 1178 1179. 1179 1180. 1180 1181. 1181 1182. 1182 1183. 1183 1184. 1184 1185. 1185 1186. 1186 1187. 1187 1188. 1188 1189. 1189 1190. 1190 1191. 1191 1192. 1192 1193.</pre>

Package/Method	Description	Code Example
		<pre> 13. # print(parm.requires_grad) 14. cum_loss = 0 15. for idx, (label, text) in enumerate(train_dataloader): 16. optimizer.zero_grad() 17. label, text = label.to(device), text.to(device) 18. predicted_label = model(text) 19. loss = criterion(predicted_label, label) 20. loss.backward() 21. #print(loss) torch.nn.utils.clip_grad_norm_(model.parameters(), 0. 22. optimizer.step() 23. cum_loss += loss.item() 24. print(f"Epoch {epoch}/{epochs} - Loss: {cum_loss}") 25. cum_loss_list.append(cum_loss) 26. accu_val = evaluate_no_tqdm(valid_dataloader, model) 27. acc_epoch.append(accu_val) 28. if model_path and accu_val > acc_old: 29. print(accu_val) 30. acc_old = accu_val 31. if save_dir is not None: 32. pass 33. #print("save model epoch", epoch) 34. #torch.save(model.state_dict(), model_path) 35. #save_list_to_file(lst=acc_epoch, filename=acc_dir) 36. #save_list_to_file(lst=cum_loss_list, filename=loss_dir) 37. time_end = time.time() 38. print(f"Training time: {time_end - time_start}") </pre>
DPO configuration	Copied!	<p>Involves training arguments, processing logging steps, evaluation strategies, and scheduling for model updates.</p> <p>Configuring DPO involves setting up administrative and technical measures for complying with data protection laws and regulations.</p>

Package/Method	Description	
		Copied!
DPOTraining class	Designed to equip professionals with the knowledge and skills for managing and overseeing data protection strategies in compliance with relevant laws and regulations.	<pre> 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 11. 11 12. 12 13. 13 14. 14 15. 15 16. 16 17. 17 18. 18 19. 19 20. 20 21. 21 22. 22 23. 23 24. 24 25. 25 </pre>
Retrieve and plot the training loss versus evaluation loss	Helps to retrieve log history and save it for data frame log. It also plots train and evaluation losses for epoch.	<pre> 1. tokenizer.pad_token = tokenizer.eos_token 2. # Create a DPO trainer 3. # This trainer will handle the fine-tuning of the model using the DPO technique 4. trainer = DPOTrainer(5. # The model to be fine-tuned 6. model, 7. # The reference model (not used in this case because LoRA has been used) 8. ref_model=None, 9. # The DPO training configuration 10. args=training_args, 11. # The beta parameter for the DPO loss function 12. beta=0.1, 13. # The training dataset 14. train_dataset=train_dataset, 15. # The evaluation dataset 16. eval_dataset=eval_dataset, 17. # The tokenizer for the model 18. tokenizer=tokenizer, 19. # The PEFT (Parallel Efficient Finetuning) configuration 20. peft_config=peft_config, 21. # The maximum prompt length 22. max_prompt_length=512, 23. # The maximum sequence length 24. max_length=512, 25.) </pre>
Traverse the IMDB data set	This code snippet traverses the IMDB data set by obtaining, loading, and exploring the data set.	<pre> 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 </pre> <pre> 1. # Retrieve log_history and save it to a dataframe 2. log = pd.DataFrame(trainer.state.log_history) 3. log_t = log[log['loss'].notna()] 4. log_e = log[log['eval_loss'].notna()] 5. # Plot train and evaluation losses 6. plt.plot(log_t["epoch"], log_t["loss"], label = "train_loss") 7. plt.plot(log_e["epoch"], log_e["eval_loss"], label = "eval_loss") 8. plt.legend() 9. plt.show() </pre>
	It also performs basic operations, visualizes the data, and analyzes and interprets the data set.	<pre> 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 11. 11 12. 12 13. 13 14. 14 15. 15 16. 16 17. 17 18. 18 19. 19 20. 20 </pre>

Package/Method	Description	Code Example
Iterators to train and test data sets	This code snippet indicates a path to the IMDB data set directory by combining temporary and subdirectory names. This code sets up the training and testing data iterators, retrieves the starting index of the training data, and prints the items from the training data set at indices.	<pre> 1. class IMBDataset(Dataset): 2. def __init__(self, root_dir, train=True): 3. """ 4. root_dir: The base directory of the IMDB dataset. 5. train: A boolean flag indicating whether to use training or test data. 6. """ 7. self.root_dir = os.path.join(root_dir, "train" if train else "test") 8. self.neg_files = [os.path.join(self.root_dir, "neg", f) for f in os.listdir(os.path.join(self.root_dir, "neg"))] 9. self.pos_files = [os.path.join(self.root_dir, "pos", f) for f in os.listdir(os.path.join(self.root_dir, "pos"))] 10. self.files = self.neg_files + self.pos_files 11. self.labels = [0] * len(self.neg_files) + [1] * len(self.pos_files) 12. self.pos_inx=len(self.pos_files) 13. def __len__(self): 14. return len(self.files) 15. def __getitem__(self, idx): 16. file_path = self.files[idx] 17. label = self.labels[idx] 18. with open(file_path, 'r', encoding='utf-8') as file: 19. content = file.read() 20. return label, content </pre> <p>Copied!</p>
yield_tokens function	Generates tokens from the collection of text data samples. The code snippet processes each text in 'data_iter' through the tokenizer and yields tokens to generate efficient, on-the-fly token generation suitable for tasks such as training machine learning models.	<pre> 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 1. root_dir = tempdir.name + '/' + 'imdb_dataset' 2. train_iter = IMBDataset(root_dir=root_dir, train=True) # For 3. training data 4. test_iter = IMBDataset(root_dir=root_dir, train=False) # For test 5. data 6. start=train_iter.pos_inx 7. for i in range(-10,10): 8. print(train_iter[start+i]) </pre> <p>Copied!</p>
Load pretrained model and its evaluation on test data	This code snippet helps download a pretrained model from URL, loads it into a specific architecture, and evaluates it on a test data set for assessing its performance.	<pre> 1. 1 2. 2 3. 3 4. 4 5. 5 1. tokenizer = get_tokenizer("basic_english") 2. def yield_tokens(data_iter): 3. """Yield tokens for each data sample.""" 4. for _, text in data_iter: 5. yield tokenizer(text) </pre> <p>Copied!</p>
Loading the Hugging Face model	This code snippet initiates a tokenizer using a pretrained 'bert-base-cased' model. It also downloads a pretrained model for the masked language model (MLM) task, and how to load the model configurations from a pretrained model.	<pre> 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 1. urlopened = urlopen('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/q 2. model_ = Net(vocab_size=vocab_size, num_class=2).to(device) 3. model_.load_state_dict(torch.load(io.BytesIO(urlopened.read()), map_location=device)) 4. evaluate(test_dataloader, model_) </pre> <p>Copied!</p>
Training a BERT model for MLM task	This code snippet trains the model with the specified parameters and data set. However, ensure that the 'SFTTrainer' is the appropriate trainer class for the task and the	<pre> 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 1. # Instantiate a tokenizer using the BERT base cased model 2. tokenizer = AutoTokenizer.from_pretrained("bert-base-cased") 3. # Download pretrained model from huggingface.co and cache. 4. model = BertForMaskedLM.from_pretrained('bert-base-cased') 5. # You can also start training from scratch by loading the model 6. configuration 7. # config = 8. AutoConfig.from_pretrained("google-bert/bert-base-cased") 9. # model = BertForMaskedLM.from_config(config) </pre> <p>Copied!</p>

Package/Method	Description	Code Example
	model is properly defined for training.	<pre> 8. 8 9. 9 10. 10 11. 11 12. 12 13. 13 14. 14 15. 15 16. 16 17. 17 1. training_args = TrainingArguments(2. output_dir='./trained_model', # Specify the output directory for the trained model 3. overwrite_output_dir=True, 4. do_eval=False, 5. learning_rate=5e-5, 6. num_train_epochs=1, # Specify the number of training epochs 7. per_device_train_batch_size=2, # Set the batch size for training 8. save_total_limit=2, # Limit the total number of saved checkpoints 9. logging_steps = 20 10.) 11. dataset = load_dataset("imdb", split="train") 12. trainer = SFTTrainer(13. model, 14. args=training_args, 15. train_dataset=dataset, 16. dataset_text_field="text", 17.) </pre>
		<div style="border: 1px solid #ccc; padding: 2px;">Copied!</div>
Load the model and tokenizer	Useful for tasks where you need to quickly classify the sentiment of a piece of text with a pretrained, efficient transformer model.	<pre> 1. 1 2. 2 1. tokenizer = DistilBertTokenizer.from_pretrained("distilbert-base-uncased-finetuned-sst-2-" 2. model = DistilBertForSequenceClassification.from_pretrained("distilbert-base-uncased-fine </pre>
		<div style="border: 1px solid #ccc; padding: 2px;">Copied!</div>
torch.no_grad()	The torch.no_grad() context manager disables gradient calculation. This reduces memory consumption and speeds up computation, as gradients are unnecessary for inference (i.e., when you are not training the model). The **inputs syntax is used to unpack a dictionary of keyword arguments in Python.	<pre> 1. 1 2. 2 3. 3 1. # Perform inference 2. with torch.no_grad(): 3. outputs = model(**inputs) </pre>
		<div style="border: 1px solid #ccc; padding: 2px;">Copied!</div>
Logits	The raw, unnormalized predictions of the model. Let's extract the logits from the model's outputs to perform further processing, such as determining the predicted class or calculating probabilities.	<pre> 1. 1 2. 2 1. logits = outputs.logits 2. logits.shape </pre>
		<div style="border: 1px solid #ccc; padding: 2px;">Copied!</div>
GPT-2 tokenizer	Helps to initialize the GPT-2 tokenizer using a pretrained model to handle encoding and decoding.	<pre> 1. 1 2. 2 1. # Load the tokenizer and model 2. tokenizer = GPT2Tokenizer.from_pretrained("gpt2") </pre>
		<div style="border: 1px solid #ccc; padding: 2px;">Copied!</div>
Load GPT-2 model	This code snippet initializes and loads the pretrained GPT-2 model. This code makes the GPT-2 model ready for generating text or other language tasks.	<pre> 1. 1 2. 2 1. # Load the tokenizer and model 2. model = GPT2LMHeadModel.from_pretrained("gpt2") </pre>
		<div style="border: 1px solid #ccc; padding: 2px;">Copied!</div>
Generate text	This code snippet generates text sequences based on the input and doesn't compute the gradient to generate output.	<pre> 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 11. 11 12. 12 13. 13 1. # Generate text 2. output_ids = model.generate(3. inputs.input_ids, 4. attention_mask=inputs.attention_mask, 5. pad_token_id=tokenizer.eos_token_id, 6. max_length=50, 7. num_return_sequences=1 8.) 9. output_ids 10. or </pre>

Package/Method	Description	Code Example
Decode the generated text	This code snippet decodes the text from the token IDs generated by a model. The code also decodes it into a readable string to print it.	<pre> 11. with torch.no_grad(): 12. outputs = model(**inputs) 13. outputs </pre> <p>Copied!</p> <pre> 1. 1 2. 2 3. 3 </pre> <pre> 1. # Decode the generated text 2. generated_text = tokenizer.decode(output_ids[0], skip_special_tokens=True) 3. print(generated_text) </pre> <p>Copied!</p> <pre> 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 11. 11 12. 12 </pre>
Hugging Face pipeline() function	The pipeline() function from the Hugging Face Transformers library is a high-level API designed to simplify the usage of pretrained models for various natural language processing (NLP) tasks. It abstracts the complexities of model loading, tokenization, inference, and post-processing, allowing users to perform complex NLP tasks with just a few lines of code.	<pre> 1. transformers.pipeline(2. task: str, 3. model: Optional = None, 4. config: Optional = None, 5. tokenizer: Optional = None, 6. feature_extractor: Optional = None, 7. framework: Optional = None, 8. revision: str = 'main', 9. use_fast: bool = True, 10. model_kwargs: Dict[str, Any] = None, 11. **kwargs 12.) </pre> <p>Copied!</p> <pre> 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 </pre>
expected_outputs	Tokenize instructions and the instructions_with_responses. Then, count the number of tokens in instructions, and discard the equivalent amount of tokens from the beginning of the tokenized instructions_with_responses vector. Finally, discard the final token in instructions_with_responses, corresponding to the eos token. Decode the resulting vector using the tokenizer, resulting in the expected_output.	<pre> 1. expected_outputs = [] 2. instructions_with_responses = formatting_prompts_func(test_dataset) 3. instructions = formatting_prompts_func_no_response(test_dataset) 4. for i in tqdm(range(len(instructions_with_responses))): 5. tokenized_instruction_with_response = tokenizer(instructions_with_responses[i], return_tensors="pt") 6. tokenized_instruction = tokenized_instruction_with_response['input_ids'][0] 7. expected_output = tokenizer.decode(tokenized_instruction_with_response['input_ids'][0]) 8. expected_outputs.append(expected_output) </pre> <p>Copied!</p> <pre> 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 </pre>
ListDataset	Inherits from Dataset and creates a torch Dataset from a list. This class is then used to generate a Dataset object from instructions.	<pre> 1. class ListDataset(Dataset): 2. def __init__(self, original_list): 3. self.original_list = original_list 4. def __len__(self): 5. return len(self.original_list) 6. def __getitem__(self, i): 7. return self.original_list[i] 8. instructions_torch = ListDataset(instructions) </pre> <p>Copied!</p> <pre> 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 </pre>
gen_pipeline	This code snippet takes the token IDs from the model output, decodes it from the table text, and prints the responses.	<pre> 1. gen_pipeline = pipeline("text-generation", 2. model=model, 3. tokenizer=tokenizer, 4. device=device, 5. batch_size=2, </pre>

Package/Method	Description	Code Example
torch.no_grad()	<p>This code generates text from the given input using a pipeline while optimizing resource usage by limiting input size and reducing gradient calculations.</p>	<pre> 6. max_length=50, 7. truncation=True, 8. padding=False, 9. return_full_text=False) Copied! 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 1. with torch.no_grad(): 2. # Due to resource limitation, only apply the function on 3 records using "instruction" 3. pipeline_iterator= gen_pipeline(instructions_torch[:3], 4. max_length=50, # this is set to 50 due to resource co 5. num_beams=5, 6. early_stopping=True,) 7. generated_outputs_base = [] 8. for text in pipeline_iterator: 9. generated_outputs_base.append(text[0]["generated_text"]) </pre>
load_dataset	<p>The data set is loaded using the load_dataset function from the data sets library, specifically loading the "train" split.</p>	<pre> 1. dataset_name = "imdb" 2. ds = load_dataset(dataset_name, split = "train") 3. N = 5 4. for sample in range(N): 5. print('text',ds[sample]['text']) 6. print('label',ds[sample]['label']) 7. ds = ds.rename_columns({"text": "review"}) 8. ds 9. ds = ds.filter(lambda x: len(x["review"]) > 200, batched=False) Copied! 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 11. 11 12. 12 13. 13 14. 14 15. 15 16. 16 17. 17 18. 18 19. 19 20. 20 21. 21 22. 22 23. 23 24. 24 25. 25 26. 26 27. 27 28. 28 29. 29 1. del(ds) 2. dataset_name="imdb" 3. ds = load_dataset(dataset_name, split="train") 4. ds = ds.rename_columns({"text": "review"}) 5. def build_dataset(config, dataset_name="imdb", input_min_text_length=2, input_max_text_le 6. """ 7. Build dataset for training. This builds the dataset from `load_dataset`, one should 8. customize this function to train the model on its own dataset. 9. Args: 10. dataset_name (`str`): 11. The name of the dataset to be loaded. 12. Returns: 13. dataloader (`torch.utils.data.DataLoader`): 14. The dataloader for the dataset. 15. """ 16. tokenizer = AutoTokenizer.from_pretrained(config.model_name) 17. tokenizer.pad_token = tokenizer.eos_token </pre>
about:blank		7/12

Package/Method	Description	Code Example
		<pre> 18. # load imdb with datasets 19. ds = load_dataset(dataset_name, split="train") 20. ds = ds.rename_columns({"text": "review"}) 21. ds = ds.filter(lambda x: len(x["review"]) > 200, batched=False) 22. input_size = LengthSampler(input_min_text_length, input_max_text_length) 23. def tokenize(sample): 24. sample["input_ids"] = tokenizer.encode(sample["review"][: input_size()]) 25. sample["query"] = tokenizer.decode(sample["input_ids"]) 26. return sample 27. ds = ds.map(tokenize, batched=False) 28. ds.set_format(type="torch") 29. return ds </pre>
		Copied!
	<p>Proximal policy optimization (PPO) is a reinforcement learning algorithm that is particularly well-suited for training generative models, including those used for chatbots. It helps address specific challenges in training these models, such as maintaining coherent and contextually appropriate dialogues. It improves policy gradient methods for chatbots by using a clipped objective function, which ensures gradual and stable policy updates. Specified configuration and components. config: Configuration settings for PPO training, such as learning rate and model name. model: The primary model to be fine-tuned using PPO. tokenizer: Tokenizer corresponding to the model, used for processing input text. dataset: Data set to be used for training, providing the input data for the model. data_collator: Data collator to handle batching and formatting of the input data.</p> <p>Initialized with LengthSampler(output_min_length, output_max_length). This object is used to sample output lengths for the generated sequences, ensuring they fall within the specified minimum and maximum length range. By varying the lengths, we can produce more diverse and natural outputs from the language model, preventing the generation of overly short or excessively long sequences and enhancing the overall quality of the responses.</p> <p>Set the max_new_tokens parameter in the generation_kwarg dictionary to the value of gen_len, which was sampled from output_length_sampler. This ensures that the maximum number of new tokens generated by the language model is within the desired length range, promoting more controlled and appropriately lengthened responses.</p>	<pre> 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 </pre>
Initialize PPOTrainer		<pre> 1. ppo_trainer = PPOTrainer(config, model, ref_model, tokenizer, dataset=dataset, data_collator=data_collator) 2. print("ppo_trainer object ", ppo_trainer) 3. device = ppo_trainer.accelerator.device 4. if ppo_trainer.accelerator.num_processes == 1: 5. device = 0 if torch.cuda.is_available() else "cpu" 6. print(device) </pre>
		Copied!
output_length_sampler		<pre> 1. 1 2. 2 3. 3 </pre>
		<pre> 1. output_min_length = 4 2. output_max_length = 16 3. output_length_sampler = LengthSampler(output_min_length, output_max_length) </pre>
		Copied!
max_new_tokens		<pre> 1. 1 2. 2 </pre>
		<pre> 1. generation_kwarg["max_new_tokens"] = gen_len 2. generation_kwarg </pre>
		Copied!
Text generation function	Tokenizes input text, generates a response, and decodes it.	<pre> 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 </pre>
		<pre> 1. gen_kwarg = {"min_length": -1, "top_k": 0.0, "top_p": 1.0, "do_sample": True, "pad_token_id": 0} 2. def generate_some_text(input_text, my_model): 3. # Tokenize the input text 4. input_ids = tokenizer(input_text, return_tensors='pt').input_ids.to(device) 5. generated_ids = my_model.generate(input_ids, **gen_kwarg) 6. # Decode the generated text 7. generated_text_ = tokenizer.decode(generated_ids[0], skip_special_tokens=True) 8. return generated_text_ </pre>

Package/Method	Description	Code Example
Generate text with PPO model	Generate text using the PPO-trained model.	<p>Copied!</p> <pre> 1. 1 2. 2 3. 3 4. input_text = "Once upon a time in a land far" 5. generated_text=generate_some_text(input_text,model_1) 6. generated_text </pre>
Sentiment analysis	Analyze the sentiment of the generated text using sentiment_pipe.	<p>Copied!</p> <pre> 1. 1 2. 2 3. pipe_outputs = sentiment_pipe(generated_text, **sent_kwargs) 4. pipe_outputs </pre>
Generate text with reference model	Generate text using the reference model.	<p>Copied!</p> <pre> 1. 1 2. 2 3. generated_text = generate_some_text(input_text,ref_model) 4. generated_text </pre>
compare_models_on_dataset	This code snippet initializes the generation parameters, prepares the data set, and converts queries into tensors for the model input. It also generates a model, helps decode the text, and uses sentiment analysis to compute sentiment scores for concentrated texts before and after applying the model. This code also compiles results, stores original queries, and converts the dictionary to a pandas data frame for easy analysis.	<p>Copied!</p> <pre> 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 11. 11 12. 12 13. 13 14. 14 15. 15 16. 16 17. 17 18. 18 19. 19 20. 20 21. 21 22. 22 23. 23 24. 24 25. 25 26. 26 27. 27 28. 28 29. 29 30. 30 31. 31 32. 32 33. 33 34. 34 35. 35 36. 36 1. def compare_models_on_dataset(model, ref_model, dataset, tokenizer, sentiment_pipe, sent_ 2. gen_kwargs = { 3. "min_length": -1, 4. "top_k": 0.0, 5. "top_p": 1.0, 6. "do_sample": True, 7. "pad_token_id": tokenizer.eos_token_id 8. } 9. bs = 16 10. game_data = dict() 11. dataset.set_format("pandas") 12. df_batch = dataset[:].sample(bs) 13. game_data["query"] = df_batch["query"].tolist() 14. query_tensors = df_batch["input_ids"].tolist() 15. response_tensors_ref, response_tensors = [], [] 16. for i in range(bs): 17. gen_len = output_length_sampler() 18. output = ref_model.generate(19. torch.tensor(query_tensors[i]).unsqueeze(dim=0).to(device), 20. max_new_tokens=gen_len, 21. **gen_kwargs 22.).squeeze()[-gen_len:] 23. response_tensors_ref.append(output) 24. output = model.generate(torch.tensor(query_tensors[i]).unsqueeze(dim=0).to(device), 25. max_new_tokens=gen_len, 26. **gen_kwargs 27.).squeeze()[-gen_len:] 28. response_tensors.append(output) 29. game_data["response (before)"] = [tokenizer.decode(response_tensors_ref[i]) for i in 30. game_data["response (after)"] = [tokenizer.decode(response_tensors[i]) for i in range 31. texts_before = [q + r for q, r in zip(game_data["query"], game_data["response (before)"]), 32. game_data["rewards (before)"] = [output[1]["score"] for output in sentiment_pipe(text 33. texts_after = [q + r for q, r in zip(game_data["query"], game_data["response (after)"]) </pre>

Package/Method	Description	Code Example
Tokenizing data	This code snippet defines a function 'compare_models_on_dataset' for comparing the performance of two models by initializing generation parameters and setting the batch size, preparing the data set in the pandas format, and sampling the batch queries.	<pre> 34. game_data["rewards (after)"] = [output[1]["score"] for output in sentiment_pipe(texts) 35. df_results = pd.DataFrame(game_data) 36. return df_results </pre> <p>Copied!</p> <pre> 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 </pre> <pre> 1. # Instantiate a tokenizer using the BERT base cased model 2. tokenizer = AutoTokenizer.from_pretrained("bert-base-cased") 3. # Define a function to tokenize examples 4. def tokenize_function(examples): 5. # Tokenize the text using the tokenizer 6. # Apply padding to ensure all sequences have the same length 7. # Apply truncation to limit the maximum sequence length 8. return tokenizer(examples["text"], padding="max_length", truncation=True) 9. # Apply the tokenize function to the dataset in batches 10. tokenized_datasets = dataset.map(tokenize_function, batched=True) </pre>
Training loop	The train_model function trains a model using a set of training data provided through a dataloader. It begins by setting up a progress bar to help monitor the training progress visually. The model is switched to training mode, which is necessary for certain model behaviors like dropout to work correctly during training. The function processes the data in batches for each epoch, which involves several steps for each batch: transferring the data to the correct device (like a GPU), running the data through the model to get outputs and calculate loss, updating the model's parameters using the calculated gradients, adjusting the learning rate, and clearing the old gradients.	<p>Copied!</p> <pre> 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 11. 11 12. 12 13. 13 14. 14 15. 15 16. 16 17. 17 18. 18 19. 19 20. 20 21. 21 22. 22 23. 23 24. 24 25. 25 26. 26 27. 27 28. 28 29. 29 30. 30 31. 31 32. 32 33. 33 34. 34 35. 35 </pre> <pre> 1. def train_model(model,tr_dataloader): 2. # Create a progress bar to track the training progress 3. progress_bar = tqdm(range(num_training_steps)) 4. # Set the model in training mode 5. model.train() 6. tr_losses=[] 7. # Training loop 8. for epoch in range(num_epochs): 9. total_loss = 0 10. # Iterate over the training data batches 11. for batch in tr_dataloader: 12. # Move the batch to the appropriate device 13. batch = {k: v.to(device) for k, v in batch.items()} 14. # Forward pass through the model 15. outputs = model(**batch) 16. # Compute the loss 17. loss = outputs.loss 18. # Backward pass (compute gradients) 19. loss.backward() 20. total_loss += loss.item() 21. # Update the model parameters 22. optimizer.step() 23. # Update the learning rate scheduler 24. lr_scheduler.step() 25. # Clear the gradients 26. optimizer.zero_grad() 27. # Update the progress bar 28. progress_bar.update(1) 29. tr_losses.append(total_loss/len(tr_dataloader)) 30. #plot loss 31. plt.plot(tr_losses) </pre>

Package/Method	Description	Code Example
	<pre> 32. plt.title("Training loss") 33. plt.xlabel("Epoch") 34. plt.ylabel("Loss") 35. plt.show() </pre>	
evaluate_model function	<p>Copied!</p> <pre> 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 11. 11 12. 12 13. 13 14. 14 15. 15 16. 16 17. 17 18. 18 19. 19 20. 20 21. 21 22. 22 </pre> <p>Works similarly to the <code>train_model</code> function but is used for evaluating the model's performance instead of training it. It uses a dataloader to process data in batches, setting the model to evaluation mode to ensure accuracy in measurements and disabling gradient calculations since it's not training. The function calculates predictions for each batch, updates an accuracy metric, and finally, prints the overall accuracy after processing all batches.</p>	<pre> 1. def evaluate_model(model, evl_dataloader): 2. # Create an instance of the Accuracy metric for multiclass classification with 5 clas 3. metric = Accuracy(task="multiclass", num_classes=5).to(device) 4. # Set the model in evaluation mode 5. model.eval() 6. # Disable gradient calculation during evaluation 7. with torch.no_grad(): 8. # Iterate over the evaluation data batches 9. for batch in evl_dataloader: 10. # Move the batch to the appropriate device 11. batch = {k: v.to(device) for k, v in batch.items()} 12. # Forward pass through the model 13. outputs = model(**batch) 14. # Get the predicted class labels 15. logits = outputs.logits 16. predictions = torch.argmax(logits, dim=-1) 17. # Accumulate the predictions and labels for the metric 18. metric(predictions, batch["labels"]) 19. # Compute the accuracy 20. accuracy = metric.compute() 21. # Print the accuracy 22. print("Accuracy:", accuracy.item()) </pre>

llm_model

This code snippet defines function '`llm_model`' for generating text using the language model from the `mistral.ai` platform, specifically the '`mistral-8x7b-instruct-v01`' model. The function helps in customizing generating parameters and interacts with IBM Watson's machine learning services.

Copied!

```

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
27. 27
28. 28
29. 29
30. 30
31. 31

```

```

1. def llm_model(prompt_txt, params=None):
2.     model_id = 'mistralai/mistral-8x7b-instruct-v01'
3.     default_params = {
4.         "max_new_tokens": 256,
5.         "min_new_tokens": 0,
6.         "temperature": 0.5,
7.         "top_p": 0.2,
8.         "top_k": 1
9.     }
10.    if params:

```

Package/Method	Description	Code Example
		<pre> 11. default_params.update(params) 12. parameters = { 13. GenParams.MAX_NEW_TOKENS: default_params["max_new_tokens"], # this controls the 14. GenParams.MIN_NEW_TOKENS: default_params["min_new_tokens"], # this controls the m 15. GenParams.TEMPERATURE: default_params["temperature"], # this randomness or creati 16. GenParams.TOP_P: default_params["top_p"], 17. GenParams.TOP_K: default_params["top_k"] 18. } 19. credentials = { 20. "url": "https://us-south.ml.cloud.ibm.com" 21. } 22. project_id = "skills-network" 23. model = Model(24. model_id=model_id, 25. params=parameters, 26. credentials=credentials, 27. project_id=project_id 28.) 29. mixtral_llm = WatsonxLLM(model=model) 30. response = mixtral_llm.invoke(prompt_txt) 31. return response </pre>
RewardTrainer	The RewardTrainer orchestrates the training process, handling tasks such as batching, optimization, evaluation, and saving model checkpoints. It is particularly useful for training models that need to learn from feedback signals, improving their ability to generate high-quality responses.	<p>Copied!</p> <pre> 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 </pre> <pre> 1. # Initialize RewardTrainer 2. trainer = RewardTrainer(3. model=model, 4. args=training_args, 5. tokenizer=tokenizer, 6. train_dataset=dataset_dict['train'], 7. eval_dataset=dataset_dict['test'], 8. peft_config=peft_config, 9.) </pre> <p>Copied!</p>



Skills Network