# LAMPIRAN

## SEKOR KEPENTINGAN BERDASARKAN SELEKSI FITUR

| No | Feature | Learning Vector Quantization | Autoencoder | Hybrid (LVQ +AE) |
|---|---|---|---|---|
| 1 | Down/Up Ratio | 480129823283,243 | 2.595.847 | 10.000.480.129.823.200 |
| 2 | Fwd IAT Max | 0.00244664265568078 | 2.326.991 | 0.8710234673207503 |
| 3 | Flow IAT Std | 0.0012304287192159934 | 23.167.068 | 0.8647800784346396 |
| 4 | Fwd IAT Std | 0.0012838139144924167 | 22.595.814 | 0.8369091643898037 |
| 5 | Fwd Packets/s | 0.12297962967090416 | 19.684.954 | 0.816315162812994 |
| 6 | Fwd Packet Length Std | 0.0004381956368947958 | 22.186.394 | 0.8160500945359732 |
| 7 | Packet Length Std | 883843287,380421 | 21.592.834 | 0.7866061499296084 |
| 8 | Flow IAT Max | 0.00244607619229701 | 20.814.953 | 0.7510185955861487 |
| 9 | Flow IAT Mean | 0.001219339087982747 | 20.691.654 | 0.7437648245845038 |
| 10 | Fwd IAT Mean | 0.0013940211695744457 | 20.132 | 0.7165822860163762 |
| 11 | Fwd IAT Total | 0.022756503886439718 | 19.549.257 | 0.7094588745300554 |
| 12 | Bwd Packet Length Mean | 5734999,06665465 | 18.827.803 | 0.6514359769950698 |
| 13 | Flow Duration | 0.022754826637736623 | 1.800.391 | 0.6339169527020372 |
| 14 | Bwd IAT Std | 0.0008451028761749541 | 18.312.672 | 0.6271001978812104 |
| 15 | Bwd IAT Min | 0.0002491454335430335 | 1.811.668 | 0.6169237474652508 |
| 16 | Bwd IAT Max | 0.0008743541701132998 | 17.820.198 | 0.6030562060339744 |
| 17 | Bwd IAT Mean | 0.0008609213158650387 | 17.771.421 | 0.6006584085747761 |
| 18 | act_data_pkt_fwd | 0.01144985518886252 | 17.136.208 | 0.5801966007752578 |
| 19 | Avg Bwd Segment Size | 5734999,06665465 | 16.740.257 | 0.5493917522559951 |
| 20 | Bwd Packets/s | 1534719244,11566 | 16.423.477 | 0.5339221050240146 |
| 21 | Flow Packets/s | 0.11887498072351974 | 13.836.476 | 0.5263229744788603 |
| 22 | Bwd Packet Length Min | 144766623,85434 | 16.032.682 | 0.5148039119709085 |
| 23 | min_seg_size_forward | 0.05116024294828498 | 14.200.882 | 0.476421233806363 |
| 24 | Bwd IAT Total | 0.0007587349577403911 | 14.902.088 | 0.46029636027054555 |
| 25 | Bwd Packet Length Max | 872760,127157149 | 14.833.134 | 0.456166983523517 |
| 26 | Source IP | 177792,129482586 | 14.786.104 | 0.4538680614818845 |
| 27 | Total Length of Fwd Packets | 0.011338570540177529 | 14.395.323 | 0.4461042968680736 |
| 28 | Bwd Packet Length Std | 8947672,26136721 | 14.562.706 | 0.4429478436778206 |
| 29 | Subflow Fwd Bytes | 0.011338570540177529 | 14.254.916 | 0.43924088162683217 |
| 30 | Active Std | 9621470213,68019 | 14.367.359 | 0.43340843447947575 |
| 31 | ACK Flag Count | 0.05424328057117848 | 1.283.981 | 0.412971838396267 |
| 32 | Idle Std | 0.0006559873088735505 | 13.554.958 | 0.39434269909705105 |
| 33 | Active Max | 5366774525,37629 | 13.170.271 | 0.374887647601094 |
| 34 | Idle Mean | 0.0011451358681902673 | 13.018.427 | 0.36860490988552425 |
| 35 | Idle Min | 0.0006306771793722579 | 12.847.796 | 0.3597496156254172 |
| 36 | Inbound | 14250943,3220319 | 12.684.168 | 0.35112038397247286 |
| 37 | Average Packet Size | 0.017751305322322462 | 12.298.994 | 0.3500434746548267 |
| 38 | URG Flag Count | 6687,52819452427 | 12.460.118 | 0.3401682973576773 |
| 39 | Timestamp | 0.049355462374800714 | 11.339.527 | 0.33474658447753813 |
| 40 | CWE Flag Count | 26972,8071212145 | 12.278.223 | 0.33127683401377517 |
| 41 | Init_Win_bytes_backward | 2219,6238136172 | 12.246.566 | 0.3297293486430592 |
| 42 | Init_Win_bytes_forward | 0.000255744325580746 | 12.124.893 | 0.3240374432677652 |
| 43 | Destination Port | 0.1375512050405048 | 8.854.386 | 0.3014627115741275 |
| 44 | Idle Max | 0.0023682461749655586 | 11.583.904 | 0.29970512256733006 |
| 45 | Source Port | 0.09167195226983432 | 9.734.361 | 0.29859870042638187 |
| 46 | Max Packet Length | 9279814418,60842 | 11.033.836 | 0.27054103580173305 |
| 47 | Subflow Bwd Packets | 2259781,6832875 | 10.616.926 | 0.25006866477675943 |
| 48 | Protocol | 0.0220284015002631 | 9.961.342 | 0.24005053938058843 |
| 49 | SimillarHTTP | 8922069,80584062 | 10.044.567 | 0.22209040222689463 |
| 50 | SYN Flag Count | 84512506132,2369 | 100.178.585 | 0.2208693256717969 |
| 51 | Min Packet Length | 0.05974319877432607 | 8.769.794 | 0.21951964797781728 |
| 52 | Fwd Packet Length Min | 0.05977329302519282 | 8.621.369 | 0.2122943668433996 |
| 53 | Total Backward Packets | 2259781,6832875 | 9.826.512 | 0.21143135451026468 |
| 54 | RST Flag Count | 300041,08795561 | 9.658.974 | 0.2032416614910277 |
| 55 | Active Mean | 966387928,148361 | 9.625.627 | 0.20161252995127502 |
| 56 | Flow Bytes/s | 0.06218296185160389 | 8.345.788 | 0.20123296871805896 |
| 57 | Total Fwd Packets | 188245727178,177 | 9.413.289 | 0.19125083529487816 |
| 58 | Avg Fwd Segment Size | 0.018013786030545507 | 9.023.004 | 0.19016770596577195 |
| 59 | Subflow Fwd Packets | 188245727178,177 | 9.329.472 | 0.187153626914179 |
| 60 | Fwd Packet Length Max | 0.009412133759261412 | 8.727.862 | 0.16713883883357172 |

| | | | | |
|---|---|---|---|---|
| 61 | Active Min | 567637466,723923 | 8.813.933 | 0.1619346304760211 |
| 62 | Fwd Packet Length Mean | 0.018013786030545507 | 8.416.355 | 0.16051327760292558 |
| 63 | Fwd PSH Flags | 300041,08795561 | 8.397.798 | 0.14159235388242328 |
| 64 | Bwd Header Length | 216249,501534683 | 8.025.743 | 0.12340541186111012 |
| 65 | Packet Length Mean | 0.007498947874031523 | 77.742.767 | 0.11861210872264907 |
| 66 | Destination IP | 1053606,05990948 | 76.275.225 | 0.10393946628377781 |
| 67 | Fwd Header Length | 0.0003482693952522547 | 76.133.947 | 0.10359713380756 |
| 68 | Packet Length Variance | 34788,9123794526 | 7.561.326 | 0.10070361945531732 |
| 69 | Fwd Header Length.1 | 0.0003482693952522547 | 7.391.903 | 0.09277008419456104 |
| 70 | Total Length of Bwd Packets | 1170,00837114257 | 7.295.001 | 0.0876850187779643 |
| 71 | Subflow Bwd Bytes | 1170,00837114257 | 7.169.628 | 0.08155649155390042 |
| 72 | Fwd URG Flags | 1,83326525861441E-71 | 7.146.491 | 0.08042549341917038 |
| 73 | Flow IAT Min | 2103495711,82639 | 70.869.484 | 0.07751701270353271 |
| 74 | ECE Flag Count | 1,73031988205889E-69 | 7.058.175 | 0.0761084035038948 |
| 75 | Bwd Avg Bulk Rate | 2,64508952603107E-70 | 69.638.114 | 0.0714956745505333 |
| 76 | Bwd Avg Packets/Bulk | 1,01861814435142E-69 | 67.175.493 | 0.059457797557115555 |
| 77 | Fwd IAT Min | 2105978656,6227 | 67.083.797 | 0.05901167288177094 |
| 78 | PSH Flag Count | 1,4548923101907E-70 | 66.895.494 | 0.05808909609913826 |
| 79 | Bwd Avg Bytes/Bulk | 5,2730247706382E-71 | 6.442.067 | 0.04599156975746155 |
| 80 | Fwd Avg Bulk Rate | 6,69611266610495E-71 | 634.496 | 0.0412447527050972 |
| 81 | FIN Flag Count | 1,02615262487496E-70 | 62.876.396 | 0.03844278305768967 |
| 82 | Fwd Avg Bytes/Bulk | 5,09310930730878E-70 | 62.738.276 | 0.03776761516928673 |
| 83 | Bwd URG Flags | 1,28427714405839E-69 | 5.874.673 | 0.01825598254799843 |
| 84 | Bwd PSH Flags | 5,71747469585135E-71 | 5.811.861 | 0.015185588970780373 |
| 85 | Fwd Avg Packets/Bulk | 1,55550609249519E-69 | 55.012.054 | 1,55550609249519E-69 |

## FITUR TERPILIH PALING BERPENGARUH ANTAR ALGORITMA SELEKSI FITUR

| NO | Learning Vector Quantization | Nilai | Autoencoder | Nilai | Gabungan (LVQ-Autoencoder) | Nilai |
|---|---|---|---|---|---|---|
| 1 | Destination Port | 0.1376 | Flow IAT Std | 230.378 | Down/Up Ratio | 1.0000 |
| 2 | Fwd Packets/s | 0.1230 | Flow IAT Max | 221.148 | Fwd IAT Max | 0.8710 |
| 3 | Flow Packets/s | 0.1189 | Fwd IAT Std | 219.899 | Flow IAT Std | 0.8648 |
| 4 | Source Port | 0.0917 | Fwd IAT Max | 219.240 | Fwd IAT Std | 0.8369 |
| 5 | Flow Bytes/s | 0.0622 | Down/Up Ratio | 213.273 | Fwd Packets/s | 0.8163 |
| 6 | Fwd Packet Length Min | 0.0598 | Flow IAT Mean | 211.345 | Fwd Packet Length Std | 0.8161 |
| 7 | Min Packet Length | 0.0597 | Fwd IAT Mean | 192.952 | Packet Length Std | 0.7866 |
| 8 | ACK Flag Count | 0.0542 | Active Std | 191.260 | Flow IAT Max | 0.7510 |
| 9 | min_seg_size_forward | 0.0512 | Bwd IAT Min | 190.403 | Flow IAT Mean | 0.7438 |
| 10 | Timestamp | 0.0494 | Bwd IAT Mean | 189.859 | Fwd IAT Mean | 0.7166 |
| 11 | **Fwd IAT Total** | **0.0228** | **Fwd IAT Total** | **187.412** | **Fwd IAT Total** | **0.7095** |
| 12 | **Flow Duration** | **0.0228** | **Flow Duration** | **174.868** | Bwd Packet Length Mean | 0.6514 |
| 13 | Protocol | 0.0220 | Fwd Packet Length Std | 174.410 | **Flow Duration** | **0.6339** |
| 14 | Avg Fwd Segment Size | 0.0180 | Bwd IAT Std | 173.681 | Bwd IAT Std | 0.6271 |
| 15 | Fwd Packet Length Mean | 0.0180 | Bwd Packet Length Mean | 170.128 | Bwd IAT Min | 0.6169 |
| 16 | Average Packet Size | 0.0178 | Bwd Packet Length Min | 169.786 | Bwd IAT Max | 0.6031 |

**HASIL TRAINING DAN VALIDASI SETIAP FOLDNYA**

| No | Scenario | Fold | Accuracy | Loss | Recall | Precision | F1-Score | ROC_AUC | Training_Time_sec |
|---|---|---|---|---|---|---|---|---|---|
| 1 | RNN_Baseline | 1 | 0,99998641 | 6,37725E-05 | 1 | 0,994673768 | 0,997329773 | 0,999998844 | 5536,789679 |
| 2 | RNN_Baseline | 2 | 0,999996603 | 4,09922E-05 | 1 | 0,998663102 | 0,999331104 | 0,999999995 | 7448,56721 |
| 3 | RNN_Baseline | 3 | 0,999996603 | 2,28131E-05 | 1 | 0,998663102 | 0,999331104 | 1 | 7365,252933 |
| 4 | RNN_Baseline | 4 | 1 | 1,13599E-13 | 1 | 1 | 1 | 1 | 11148,33204 |
| 5 | RNN_Baseline | 5 | 1 | 1,41099E-13 | 1 | 1 | 1 | 1 | 9298,5369 |
| 6 | RNN_Baseline | 6 | 1 | 4,0557E-15 | 1 | 1 | 1 | 1 | 17054,29346 |
| 7 | RNN_Baseline | 7 | 1 | 1,28246E-07 | 1 | 1 | 1 | 1 | 12514,02831 |
| 8 | RNN_Baseline | 8 | 1 | 2,75984E-12 | 1 | 1 | 1 | 1 | 9703,030469 |
| 9 | RNN_Baseline | 9 | 0,999989808 | 0,000190245 | 0,998661312 | 0,997326203 | 0,997993311 | 0,999609192 | 5916,527529 |
| 10 | RNN_Baseline | 10 | 1 | 2,42108E-13 | 1 | 1 | 1 | 1 | 13572,56119 |
| 11 | RNN_LVQ_FS | 1 | 0,999666929 | 0,001722301 | 1 | 0,884023669 | 0,938442211 | 0,9999804 | 14211,92187 |
| 12 | RNN_LVQ_FS | 2 | 0,999724686 | 0,001336012 | 1 | 0,902173913 | 0,948571429 | 0,999993578 | 24842,21754 |
| 13 | RNN_LVQ_FS | 3 | 0,999731481 | 0,001686791 | 0,998661312 | 0,905339806 | 0,949713558 | 0,999983727 | 15042,9911 |
| 14 | RNN_LVQ_FS | 4 | 0,999782503 | 0,00104468 | 0,998661312 | 0,922126082 | 0,958868895 | 0,999994337 | 24829,41781 |
| 15 | RNN_LVQ_FS | 5 | 0,999741673 | 0,001360234 | 0,998661312 | 0,90864799 | 0,951530612 | 0,999988959 | 24532,37423 |
| 16 | RNN_LVQ_FS | 6 | 0,999728084 | 0,001785907 | 0,997322624 | 0,905224787 | 0,949044586 | 0,999317065 | 23210,43592 |
| 17 | RNN_LVQ_FS | 7 | 0,999738276 | 0,001347821 | 1 | 0,906553398 | 0,950986633 | 0,999988136 | 26003,49236 |
| 18 | RNN_LVQ_FS | 8 | 0,999731481 | 0,001518577 | 0,997322624 | 0,906326034 | 0,949649458 | 0,99997419 | 24532,1022 |
| 19 | RNN_LVQ_FS | 9 | 0,999707699 | 0,001550555 | 0,997322624 | 0,8986731 | 0,945431472 | 0,999986918 | 23509,94106 |
| 20 | RNN_LVQ_FS | 10 | 0,999704301 | 0,001800791 | 1 | 0,895683453 | 0,944971537 | 0,999986291 | 24449,76675 |
| 21 | RNN_AE_FS | 1 | 0,997284472 | 0,078150697 | 0,923694779 | 0,481843575 | 0,633318036 | 0,973076527 | 23513,22043 |
| 22 | RNN_AE_FS | 2 | 0,997328639 | 0,076492667 | 0,899598394 | 0,485900217 | 0,630985915 | 0,972159955 | 23677,37225 |
| 23 | RNN_AE_FS | 3 | 0,995377839 | 0,085302234 | 0,927710843 | 0,346673337 | 0,504734159 | 0,971426964 | 14434,86563 |
| 24 | RNN_AE_FS | 4 | 0,998137534 | 0,052544057 | 0,904953146 | 0,586296618 | 0,711578947 | 0,96194011 | 24096,72615 |
| 25 | RNN_AE_FS | 5 | 0,996428013 | 0,083016604 | 0,93038822 | 0,410271547 | 0,569438755 | 0,963221166 | 14491,3073 |
| 26 | RNN_AE_FS | 6 | 0,994837403 | 0,084242553 | 0,919678715 | 0,320130475 | 0,474939509 | 0,980587255 | 5963,446764 |
| 27 | RNN_AE_FS | 7 | 0,996128917 | 0,072443932 | 0,925033467 | 0,38951522 | 0,548195161 | 0,959021531 | 13713,27465 |
| 28 | RNN_AE_FS | 8 | 0,987754583 | 0,133871153 | 0,919678715 | 0,162411348 | 0,276069922 | 0,982593714 | 5103,587289 |
| 29 | RNN_AE_FS | 9 | 0,993923187 | 0,083513908 | 0,923694779 | 0,285006196 | 0,435606061 | 0,982325564 | 5632,980083 |
| 30 | RNN_AE_FS | 10 | 0,996550381 | 0,080109246 | 0,931726908 | 0,419277108 | 0,578313253 | 0,962295151 | 19571,39179 |
| 31 | RNN_Hybrid_FS | 1 | 0,988838792 | 0,059216708 | 0,9437751 | 0,178616671 | 0,300383468 | 0,986506763 | 7103,079185 |
| 32 | RNN_Hybrid_FS | 2 | 0,990001142 | 0,077599414 | 0,954484605 | 0,196906932 | 0,326465201 | 0,990759172 | 18666,41935 |
| 33 | RNN_Hybrid_FS | 3 | 0,991523743 | 0,057542291 | 0,955823293 | 0,22488189 | 0,364099949 | 0,987201336 | 15814,40163 |
| 34 | RNN_Hybrid_FS | 4 | 0,989399612 | 0,050570294 | 0,938420348 | 0,185744568 | 0,310108383 | 0,987132378 | 23210,72802 |
| 35 | RNN_Hybrid_FS | 5 | 0,992210269 | 0,063137665 | 0,957161981 | 0,240336134 | 0,384202042 | 0,989382767 | 14395,19046 |
| 36 | RNN_Hybrid_FS | 6 | 0,989447117 | 0,060754843 | 0,933065596 | 0,185767591 | 0,309846633 | 0,985812799 | 5560,982442 |
| 37 | RNN_Hybrid_FS | 7 | 0,992166102 | 0,048293807 | 0,953145917 | 0,238765929 | 0,381871816 | 0,989294051 | 21795,75817 |
| 38 | RNN_Hybrid_FS | 8 | 0,992954552 | 0,05324189 | 0,950468541 | 0,258557902 | 0,40652734 | 0,993905953 | 24542,41898 |
| 39 | RNN_Hybrid_FS | 9 | 0,987533689 | 0,073827505 | 0,950468541 | 0,163556784 | 0,27908805 | 0,988747541 | 6153,271684 |
| 40 | RNN_Hybrid_FS | 10 | 0,991343617 | 0,067019686 | 0,953145917 | 0,220843672 | 0,358599849 | 0,985280071 | 20068,42768 |

**Hasil Rata-Rata Gabungan setelah traning dan validasi**

| Scenario | Mean_Accuracy | Std_Accuracy | Mean_Accuracy (%) | Std_Accuracy (%) | Mean_Loss | Std_Loss | Mean_Recall | Std_Recall | Mean_Recall (%) | Std_Recall (%) | Mean_Precision | Std_Precision | Mean_Precision (%) | Std_Precision (%) | Mean_F1_Score | Std_F1_Score |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RNN_Baseline | 1 | 4,67074E-06 | 99,99969 | 0,00047 | 3,17951E-05 | 5,68784E-05 | 0,99987 | 0,0004 | 99,9866 | 0,04016 | 0,99893 | 0,00166 | 99,8933 | 0,16643 | 0,9994 | 0,00092 |
| RNN_LVQ_FS | 0,99973 | 2,81946E-05 | 99,97257 | 0,00282 | 0,001515367 | 0,000231077 | 0,9988 | 0,00111 | 99,8795 | 0,1112 | 0,90348 | 0,00929 | 90,3477 | 0,92875 | 0,94872 | 0,00499 |
| RNN_AE_FS | 0,99538 | 0,002803938 | 99,53751 | 0,28039 | 0,082968705 | 0,01925665 | 0,92062 | 0,00998 | 92,0616 | 0,99829 | 0,38873 | 0,11269 | 38,8733 | 11,2692 | 0,53632 | 0,11607 |
| RNN_Hybrid_FS | 0,99054 | 0,00166117 | 99,05419 | 0,16612 | 0,06112041 | 0,009090418 | 0,949 | 0,00758 | 94,8996 | 0,75834 | 0,2094 | 0,02988 | 20,9398 | 2,9883 | 0,34212 | 0,04035 |

| Scenario | Mean_F1_Score (%) | Std_F1_Score (%) | Mean_ROC_AUC | Std_ROC_AUC | Mean_ROC_AUC (%) | Std_ROC_AUC (%) | Mean_TN | Std_TN | Mean_FP | Std_FP | Mean_FN | Std_FN | Mean_TP | Std_TP | Mean_Training_Time_sec | Std_Training_Time_sec |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RNN_Baseline | 99,9399 | 0,091837276 | 0,999961 | 0,00012 | 99,99608031 | 0,011720432 | 293486 | 1,11355 | 0,8 | 1,249 | 0,1 | 0,3 | 746,9 | 0,3 | 9955,79 | 3475,24 |
| RNN_LVQ_FS | 94,8721 | 0,498511818 | 0,999919 | 0,0002 | 99,99193602 | 0,02008437 | 293407 | 8,54634 | 79,8 | 8,54166 | 0,9 | 0,83066 | 746,1 | 0,83066 | 22516,5 | 4013,35 |
| RNN_AE_FS | 53,6318 | 11,60714817 | 0,970865 | 0,0085 | 97,08647935 | 0,849767626 | 292185 | 826,574 | 1301,5 | 826,336 | 59,3 | 7,45721 | 687,7 | 7,45721 | 15019,8 | 7217,96 |
| RNN_Hybrid_FS | 34,2119 | 4,034905885 | 0,988402 | 0,00247 | 98,8402283 | 0,246622546 | 290742 | 485,729 | 2744,8 | 485,874 | 38,1 | 5,6648 | 708,9 | 5,6648 | 15731,1 | 6849,77 |

**CONFUTION MATRIX**

**LEARNING VECTOR QUATIZIATION**

| | Pred_0 | Pred_1 |
|---|---|---|
| True_0 | 2934066 | 798 |
| True_1 | 9 | 7461 |

**BASELINE**

| | Pred_0 | Pred_1 |
|---|---|---|
| True_0 | 2934856 | 8 |
| True_1 | 1 | 7469 |

**GABUNGAN (LVQ+AUTOENCODER)**

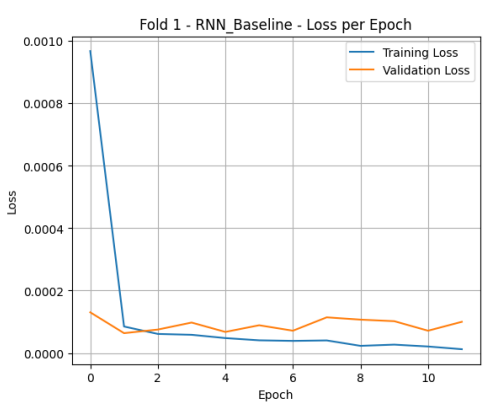
| | Pred_0 | Pred_1 |
|---|---|---|
| True_0 | 2907416 | 27448 |
| True_1 | 381 | 7089 |

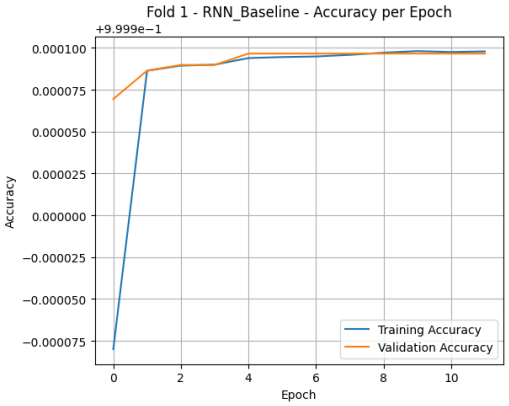
**AUTIENCODER**

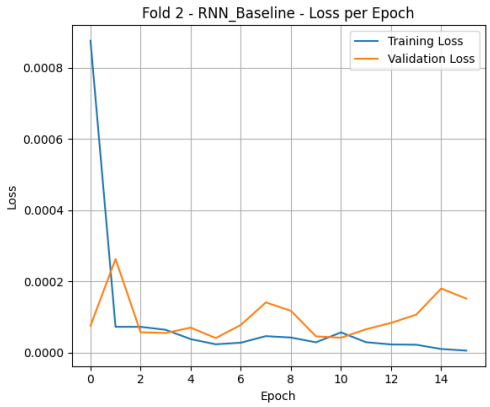
| | Pred_0 | Pred_1 |
|---|---|---|
| True_0 | 2921849 | 13015 |
| True_1 | 593 | 6877 |

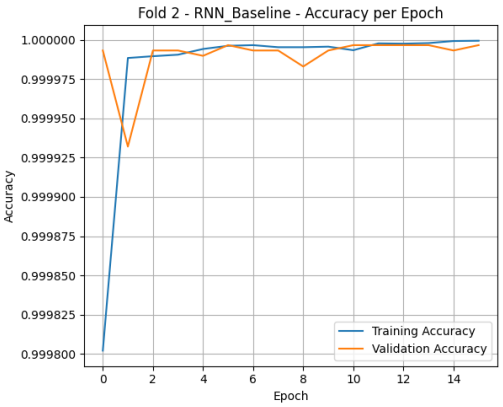
# GRAFIK LOS DAN AKURASI DAPA SETIAP FOLD BERDASARKAN SELEKSIFITUR

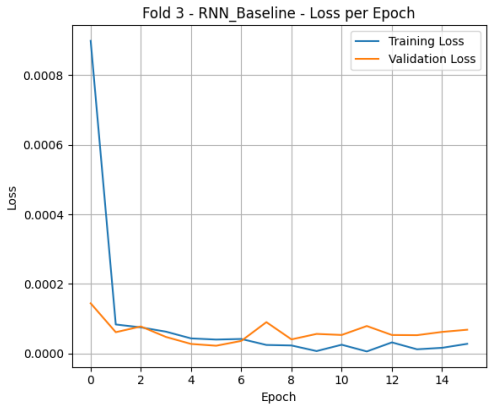
## 1. Baseline (Non Fiture Selection)

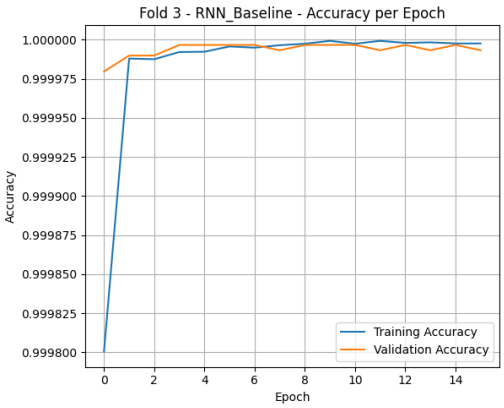


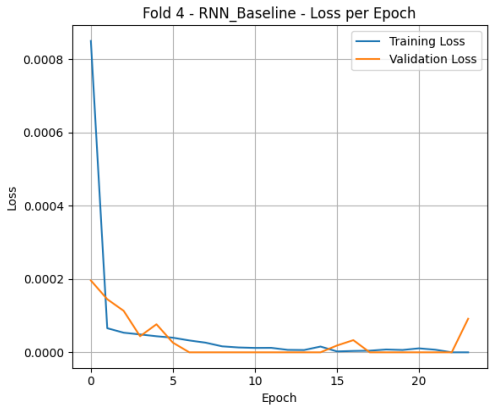



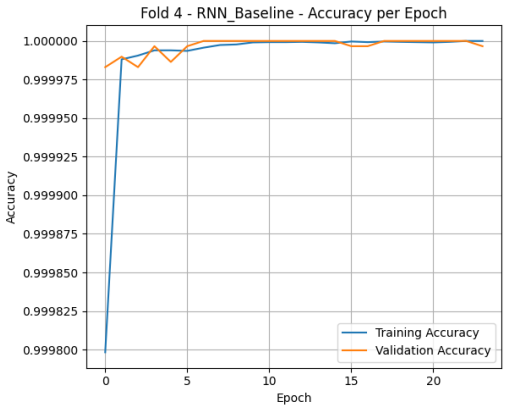



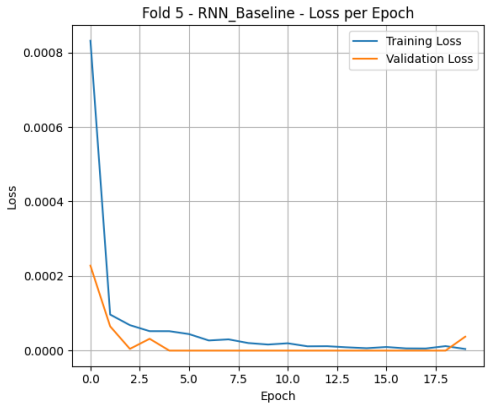



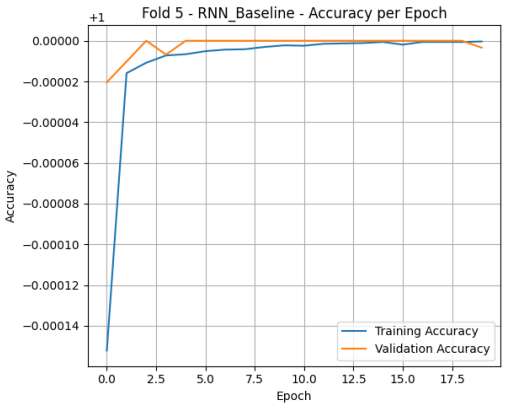



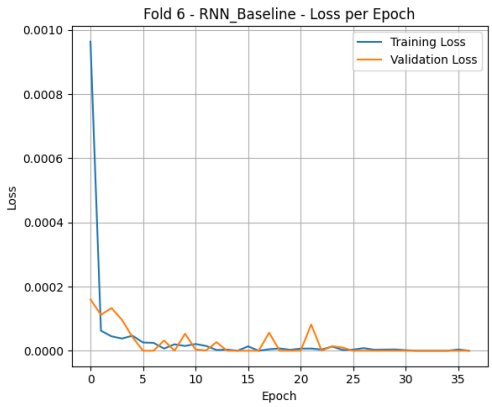



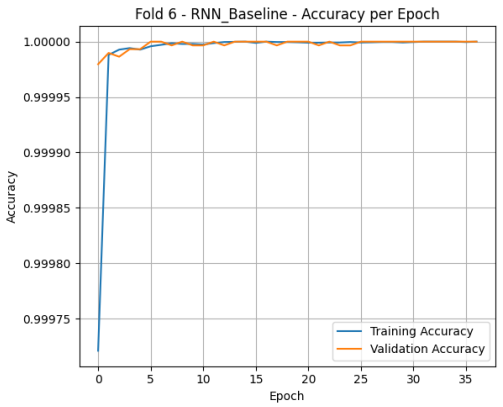

Fold 4 - RNN_Baseline - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

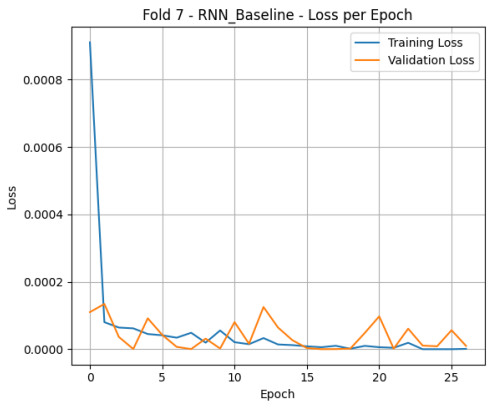

Fold 4 - RNN_Baseline - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

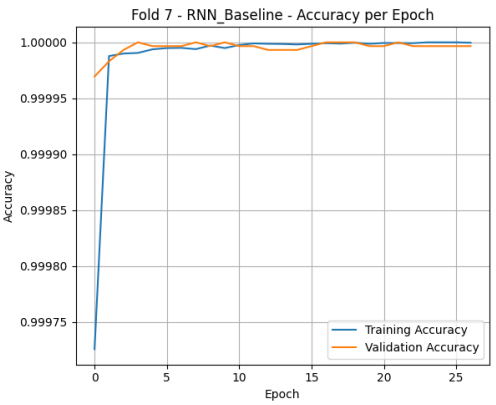

Fold 5 - RNN_Baseline - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

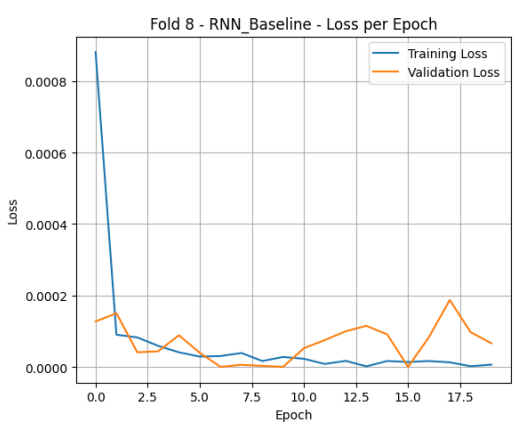

Fold 5 - RNN_Baseline - Accuracy per Epoch
+1
Training Accuracy
Validation Accuracy
Accuracy
Epoch

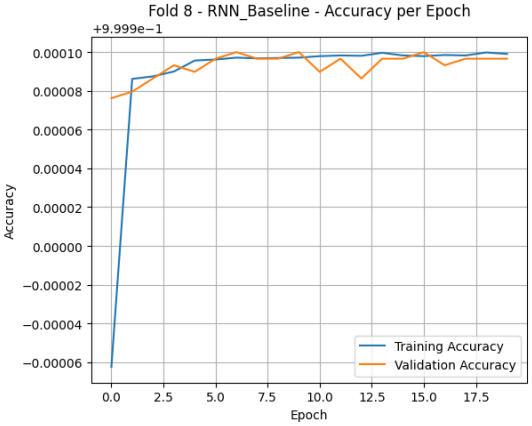

Fold 6 - RNN_Baseline - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

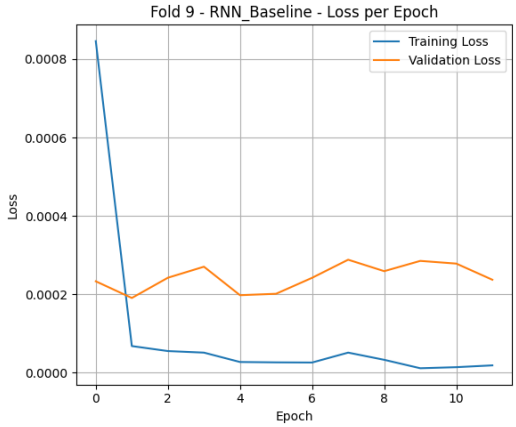

Fold 6 - RNN_Baseline - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

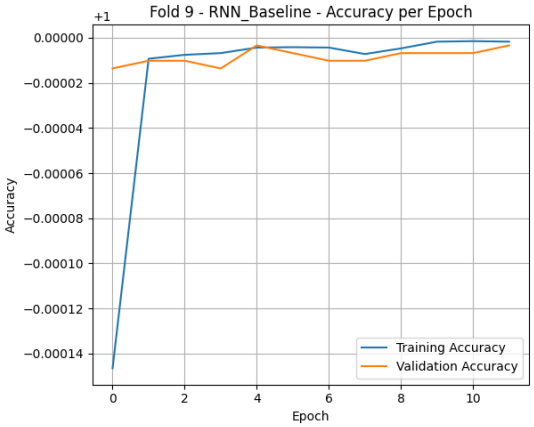

Fold 7 - RNN_Baseline - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

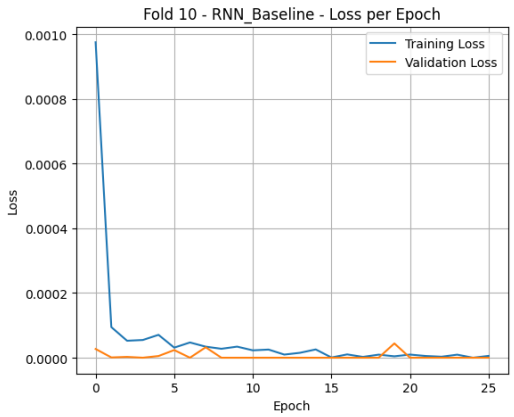

Fold 7 - RNN_Baseline - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

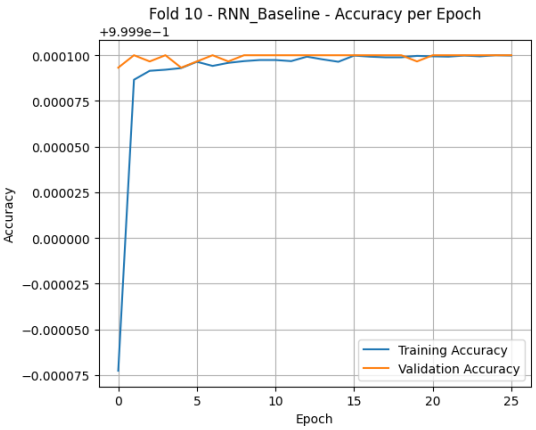

Fold 8 - RNN_Baseline - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch


Fold 8 - RNN_Baseline - Accuracy per Epoch
+9.999e−1
Training Accuracy
Validation Accuracy
Accuracy
Epoch

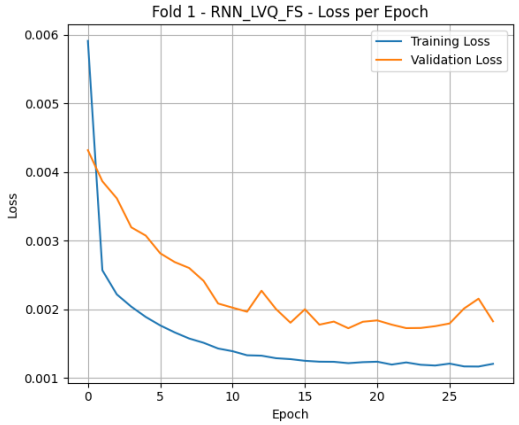

Fold 9 - RNN_Baseline - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

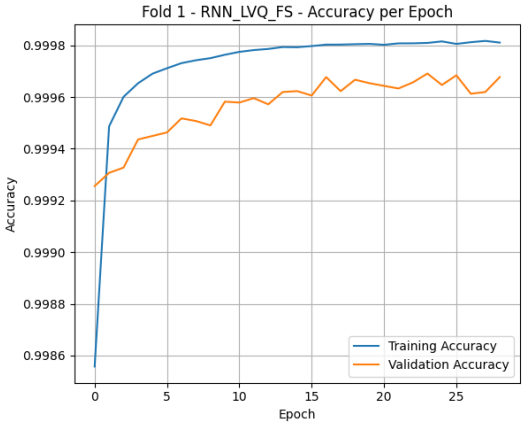

Fold 9 - RNN_Baseline - Accuracy per Epoch
+1
Training Accuracy
Validation Accuracy
Accuracy
Epoch

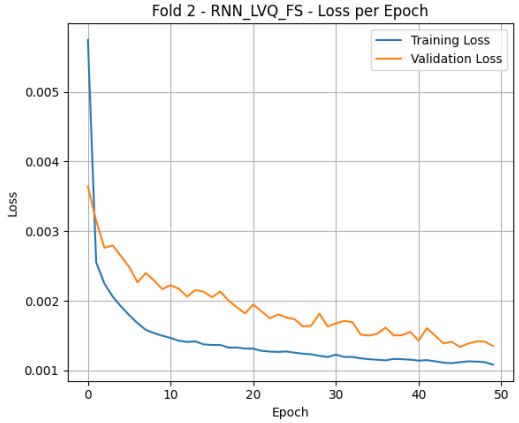

Fold 10 - RNN_Baseline - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

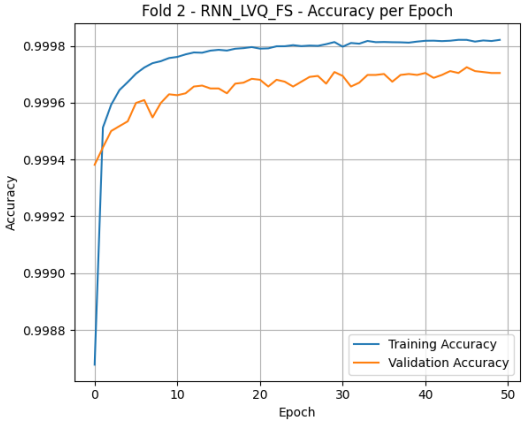

Fold 10 - RNN_Baseline - Accuracy per Epoch
+9.999e−1
Training Accuracy
Validation Accuracy
Accuracy
Epoch

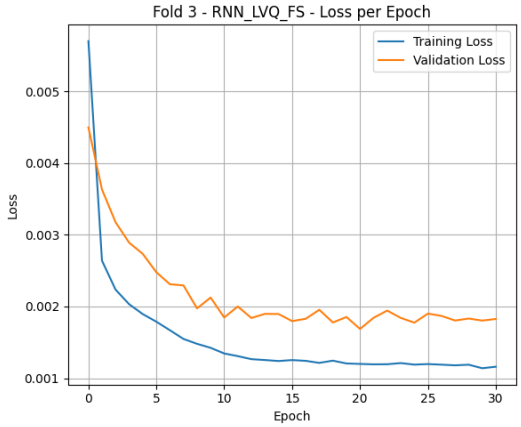
## 2. Learning Vector Quantization

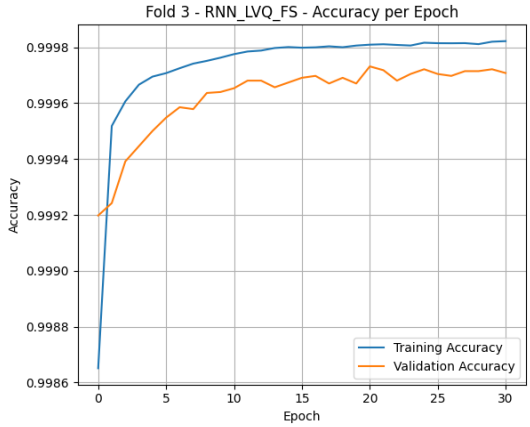


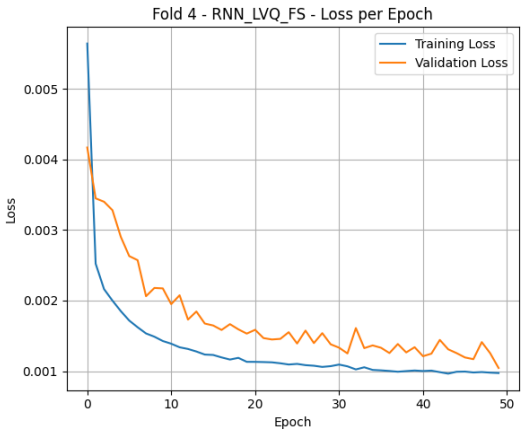



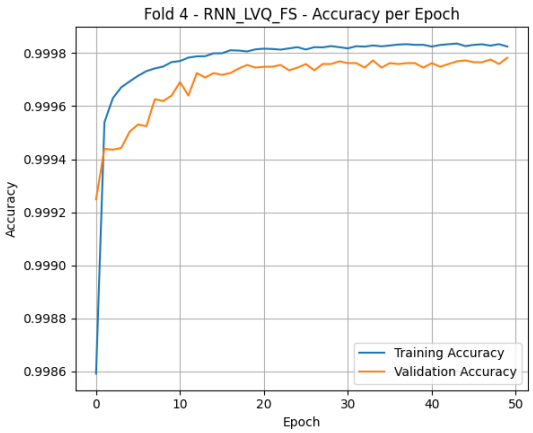



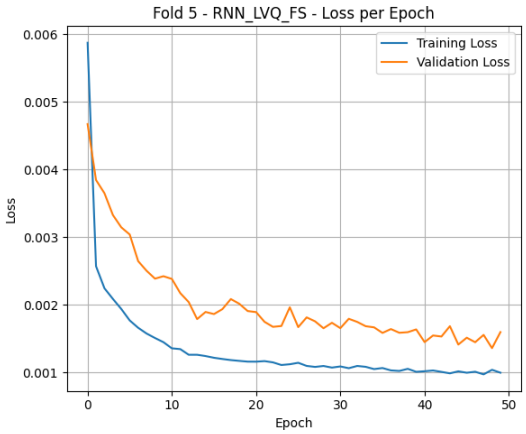



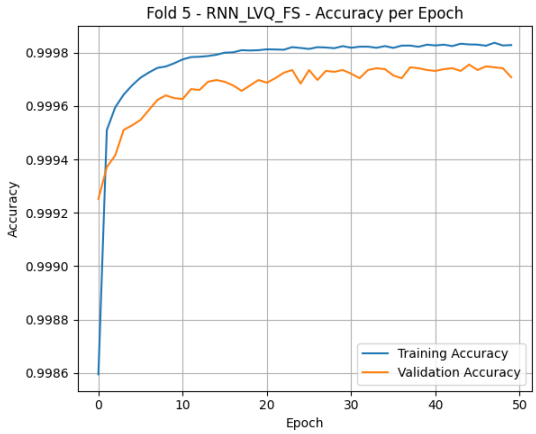



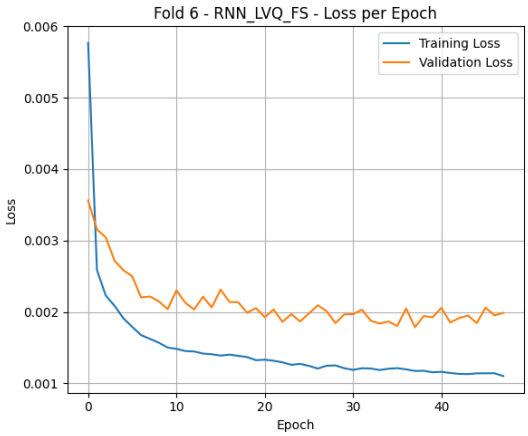



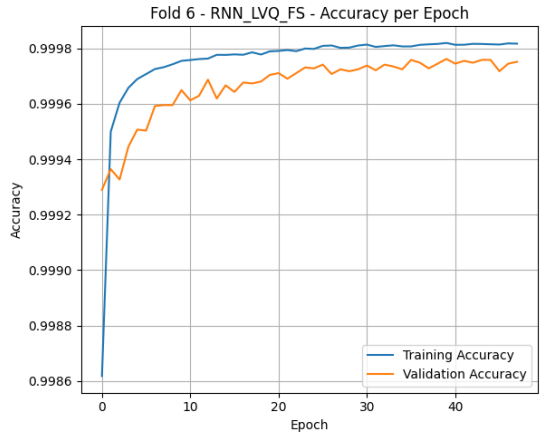

Fold 4 - RNN_LVQ_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

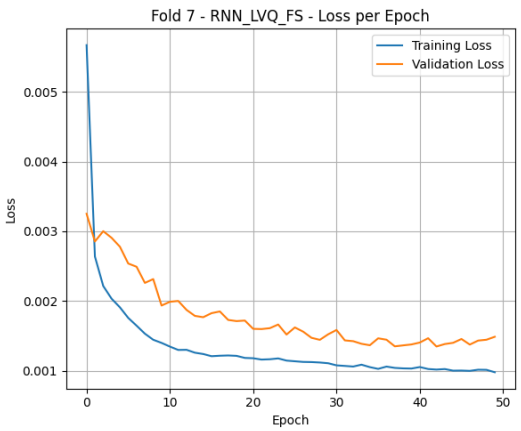

Fold 4 - RNN_LVQ_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

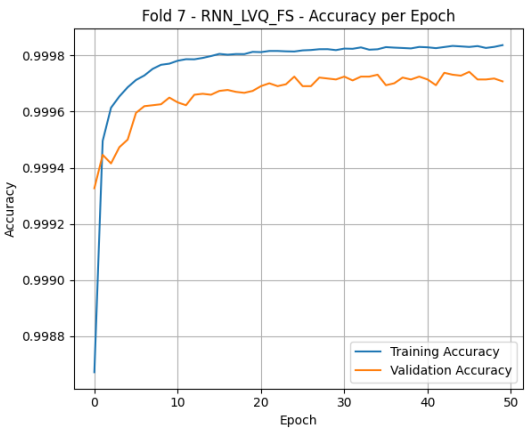

Fold 5 - RNN_LVQ_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

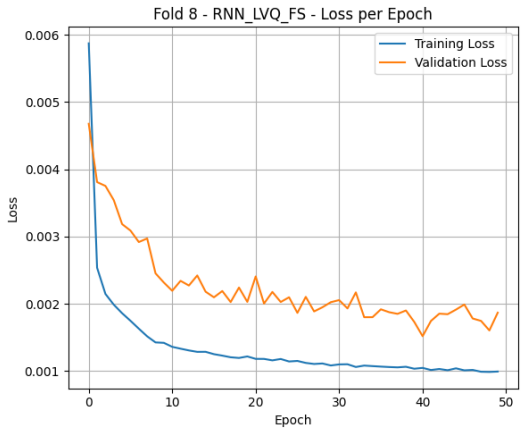

Fold 5 - RNN_LVQ_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

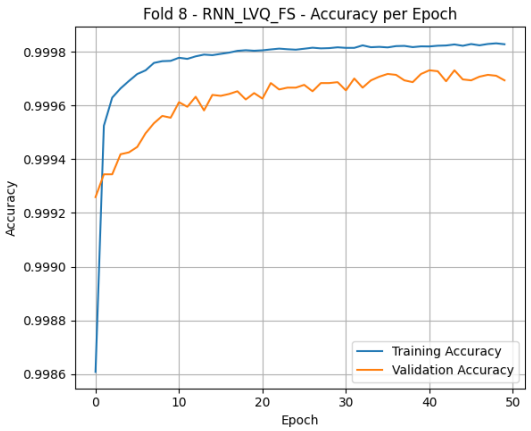

Fold 6 - RNN_LVQ_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

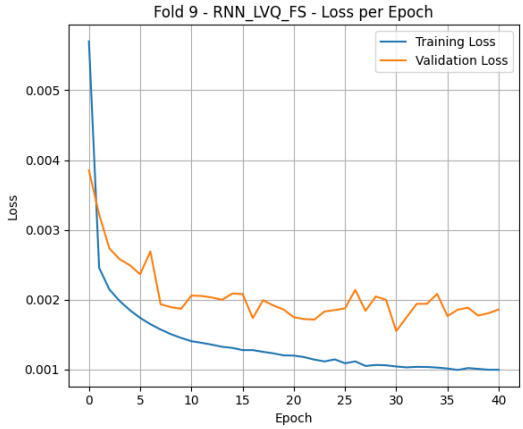

Fold 6 - RNN_LVQ_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

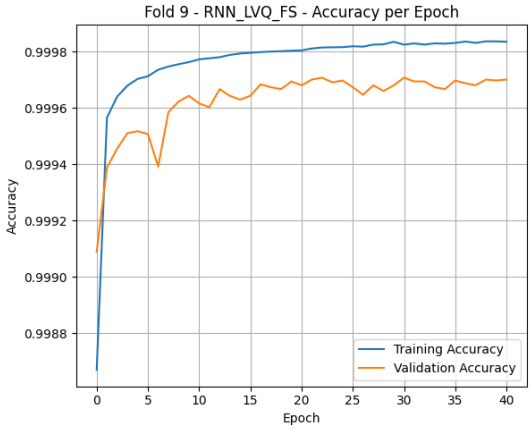

Fold 7 - RNN_LVQ_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

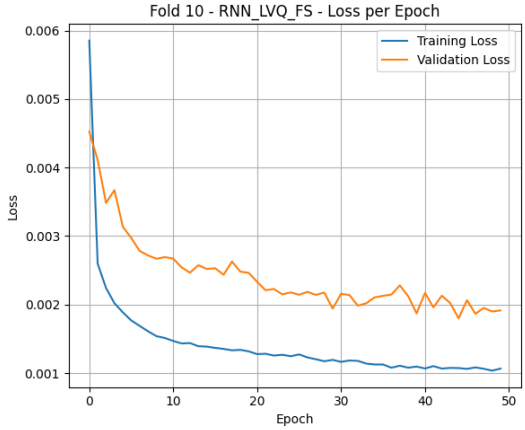

Fold 7 - RNN_LVQ_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

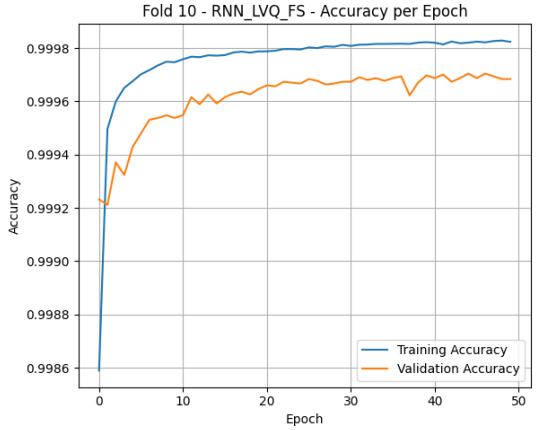

Fold 8 - RNN_LVQ_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch


Fold 8 - RNN_LVQ_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

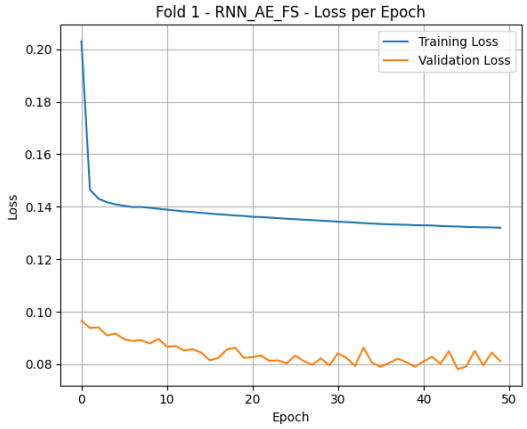

Fold 9 - RNN_LVQ_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

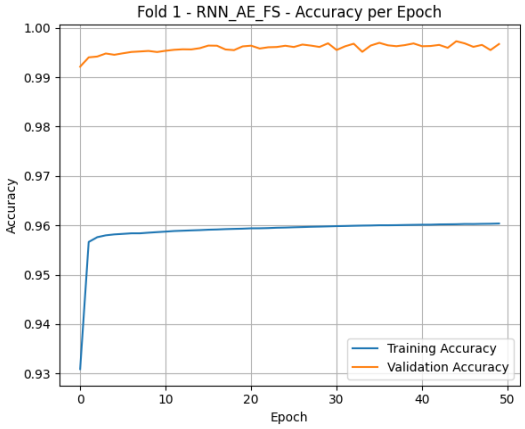

Fold 9 - RNN_LVQ_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

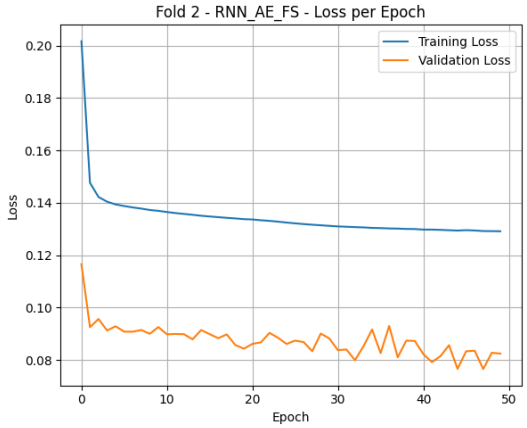

Fold 10 - RNN_LVQ_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

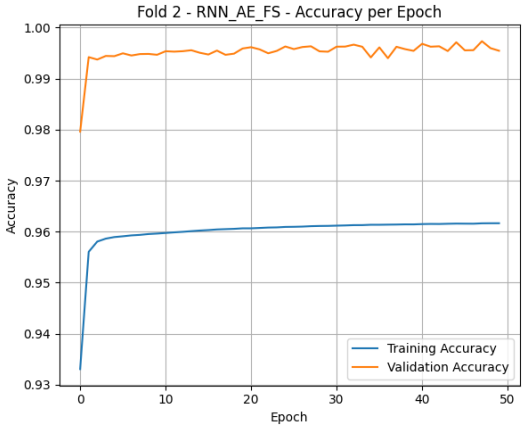

Fold 10 - RNN_LVQ_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

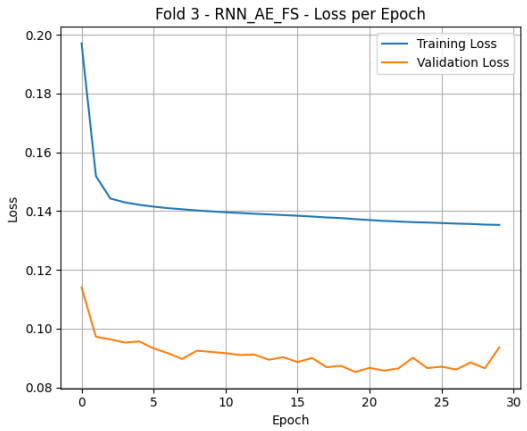
## 3. Autoencoder

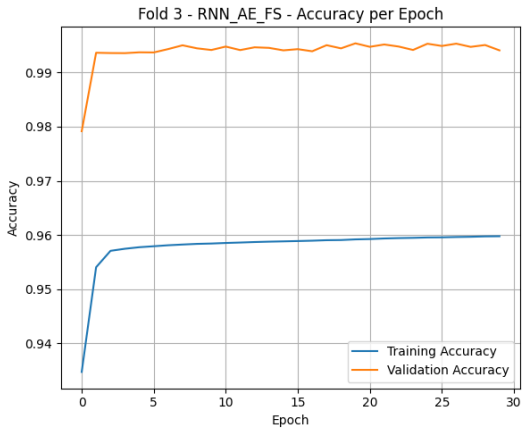


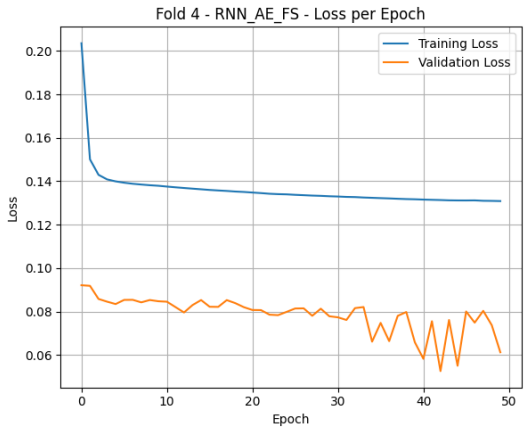



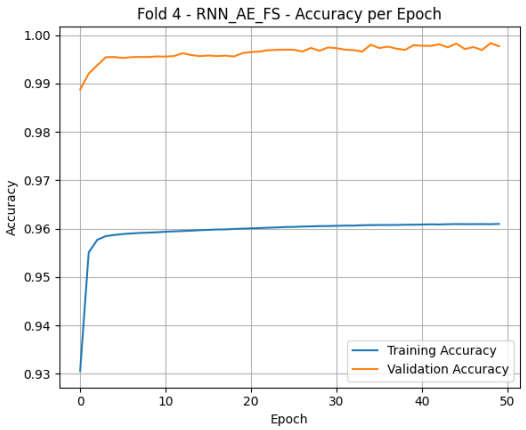



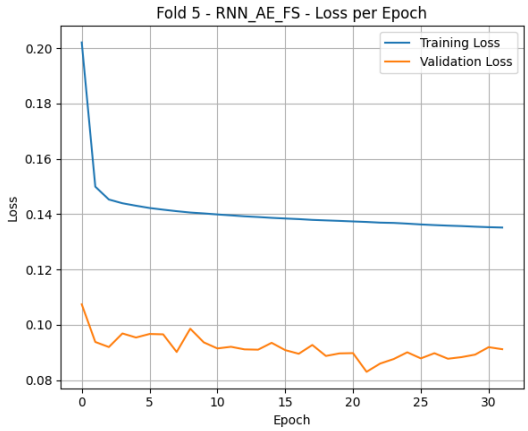



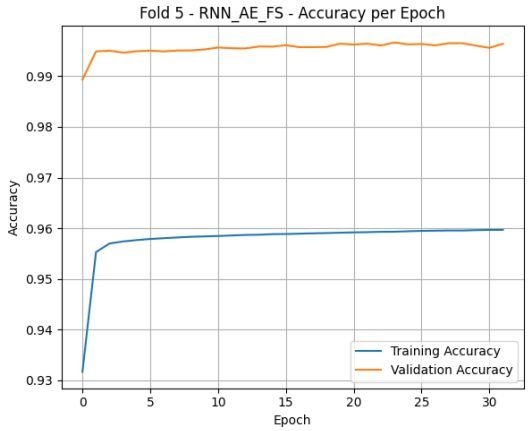



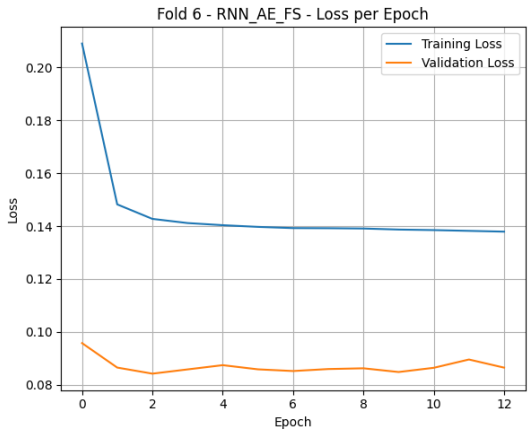



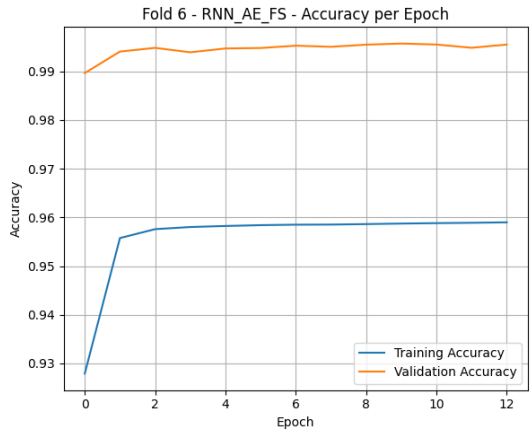

Fold 4 - RNN_AE_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

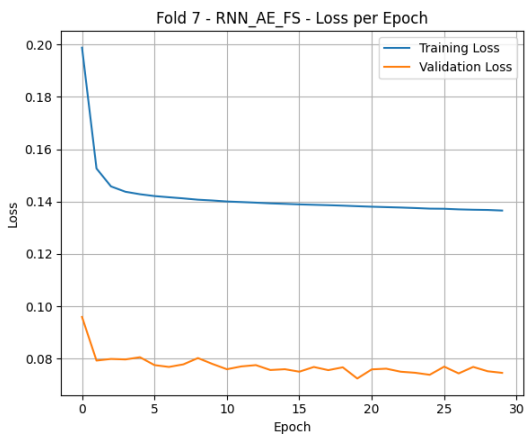

Fold 4 - RNN_AE_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

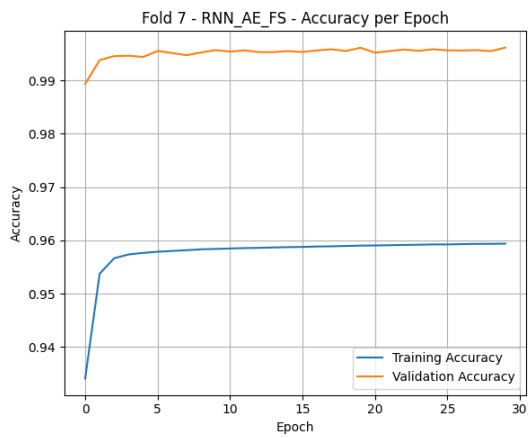

Fold 5 - RNN_AE_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

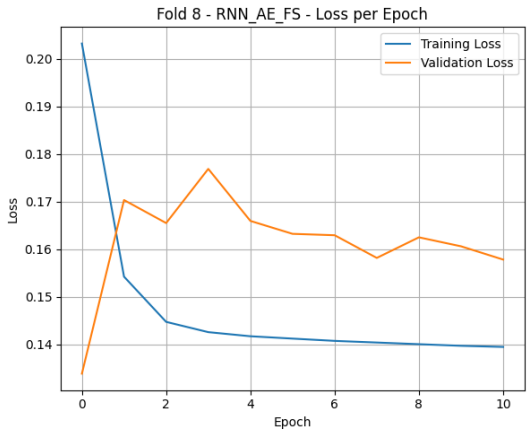

Fold 5 - RNN_AE_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

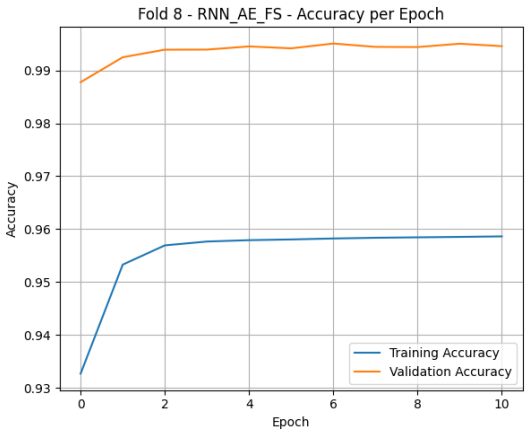

Fold 6 - RNN_AE_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

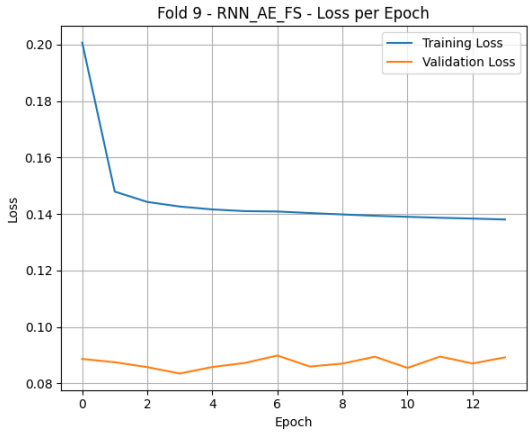

Fold 6 - RNN_AE_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

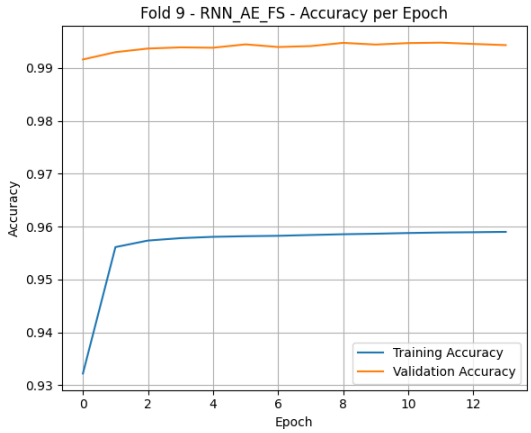

Fold 7 - RNN_AE_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

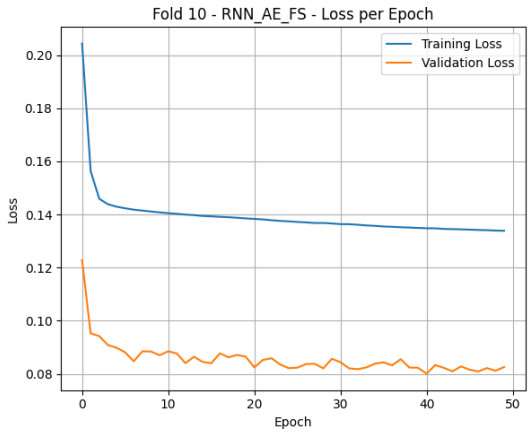

Fold 7 - RNN_AE_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

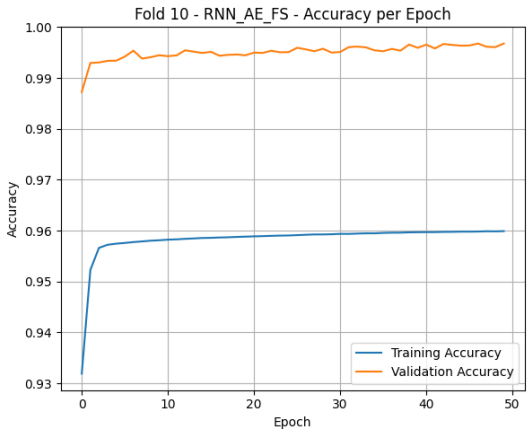

Fold 8 - RNN_AE_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch


Fold 8 - RNN_AE_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

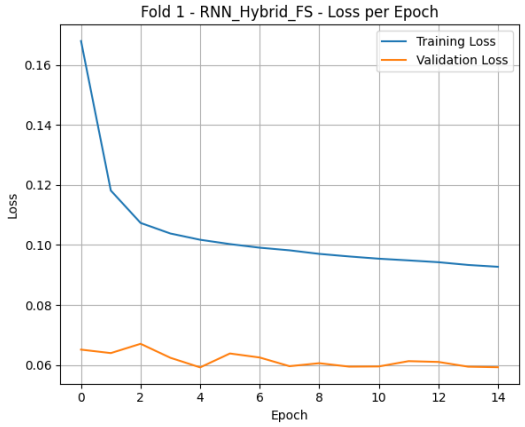

Fold 9 - RNN_AE_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

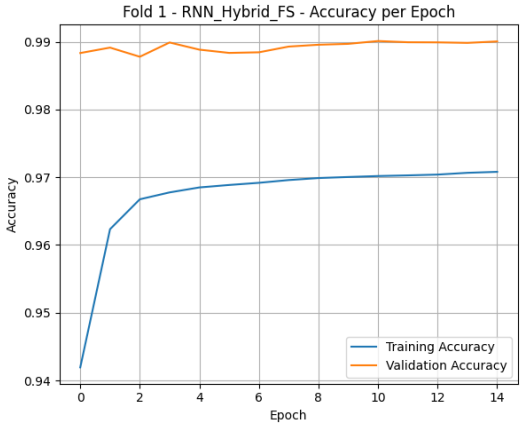

Fold 9 - RNN_AE_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

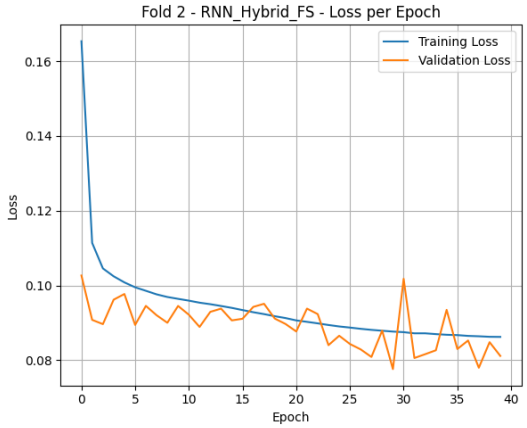

Fold 10 - RNN_AE_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

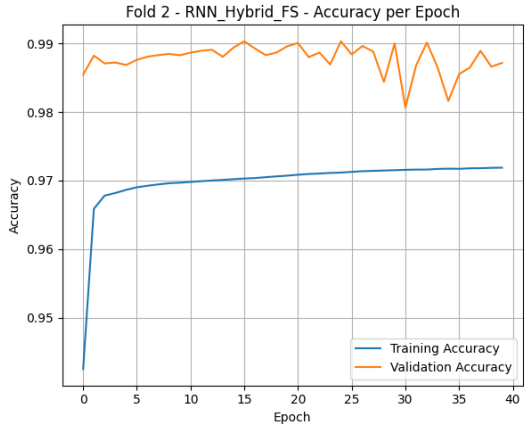

Fold 10 - RNN_AE_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

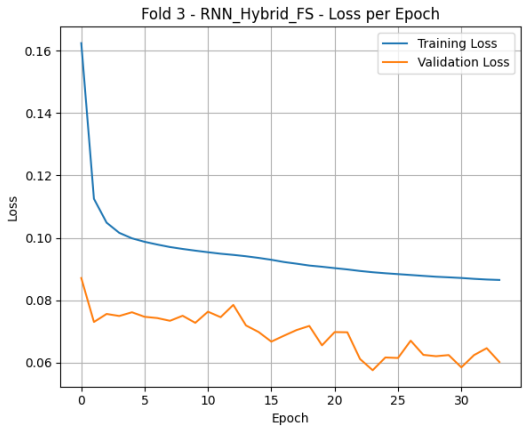
## 4. Hybrit (LVQ+AE)

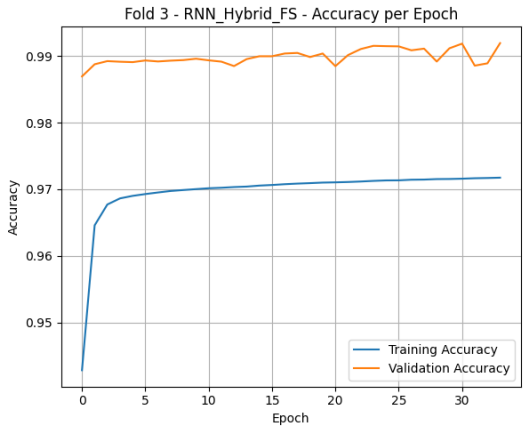


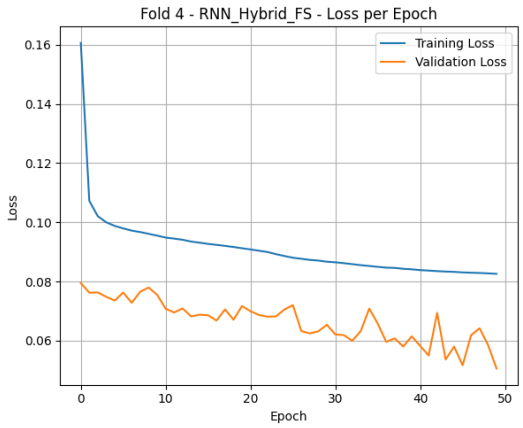



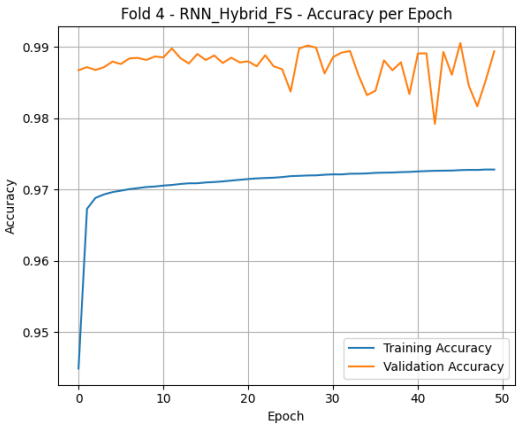



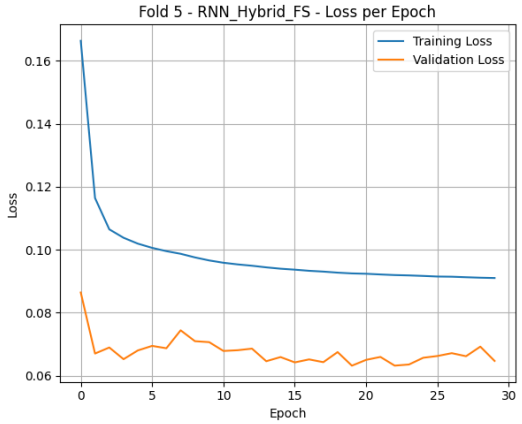



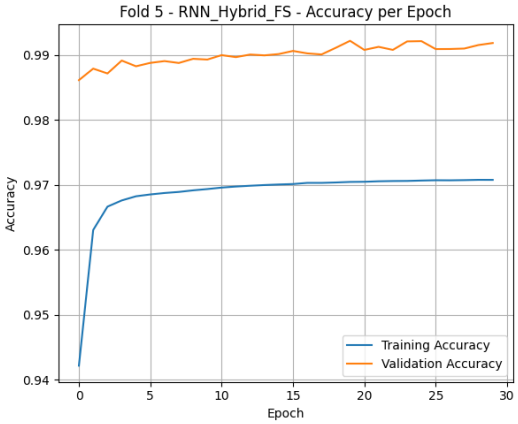



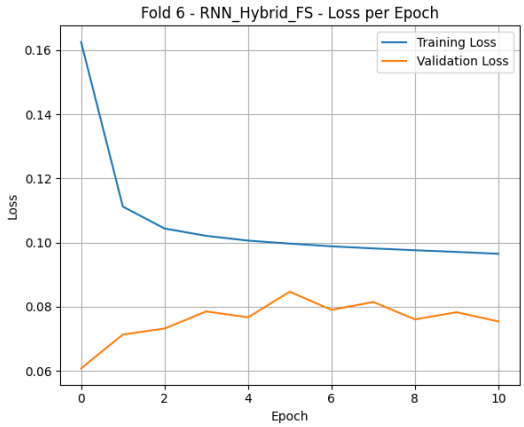



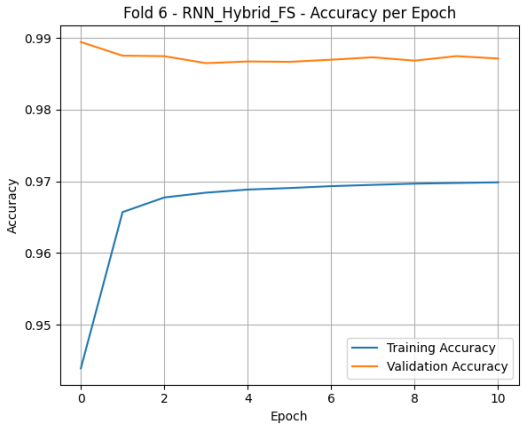

Fold 4 - RNN_Hybrid_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

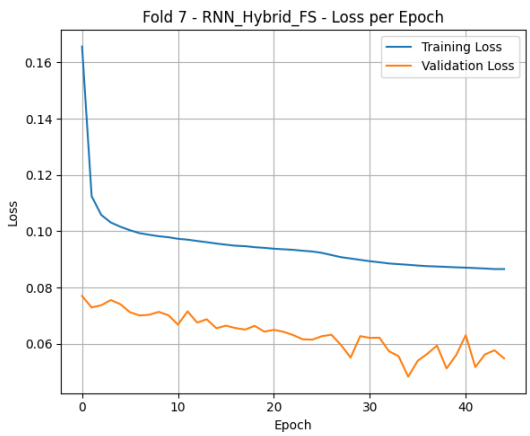

Fold 4 - RNN_Hybrid_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

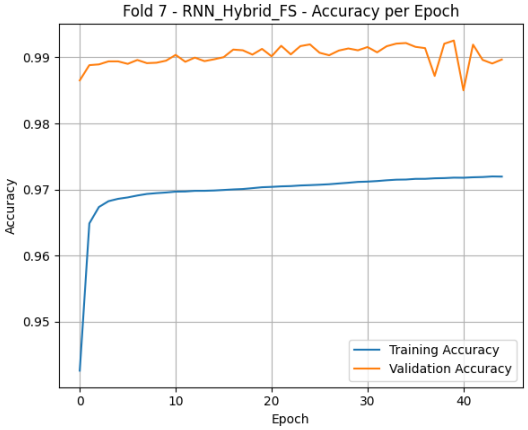

Fold 5 - RNN_Hybrid_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

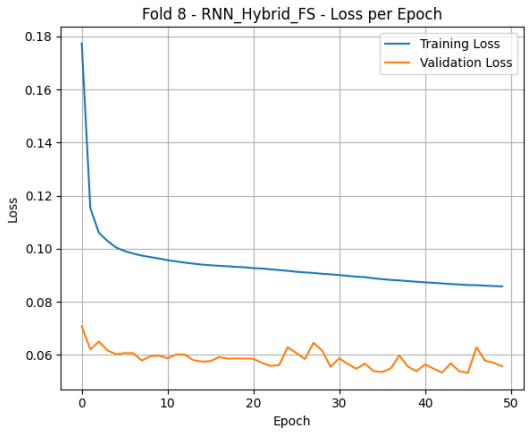

Fold 5 - RNN_Hybrid_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

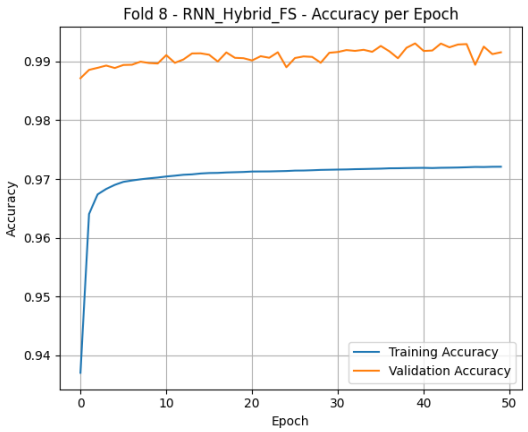

Fold 6 - RNN_Hybrid_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

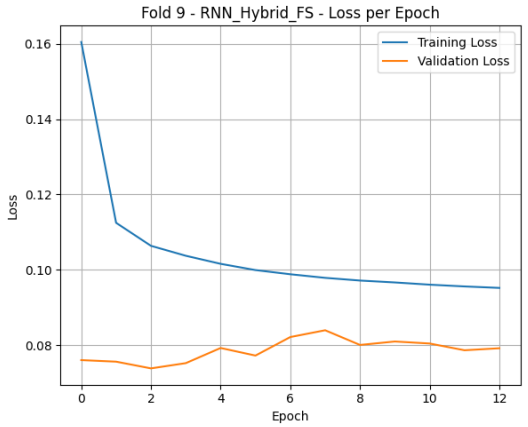

Fold 6 - RNN_Hybrid_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

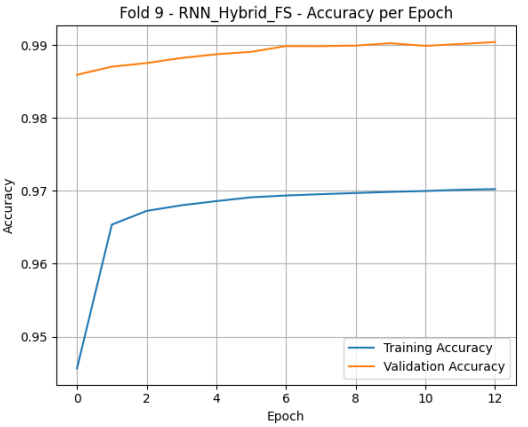

Fold 7 - RNN_Hybrid_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch

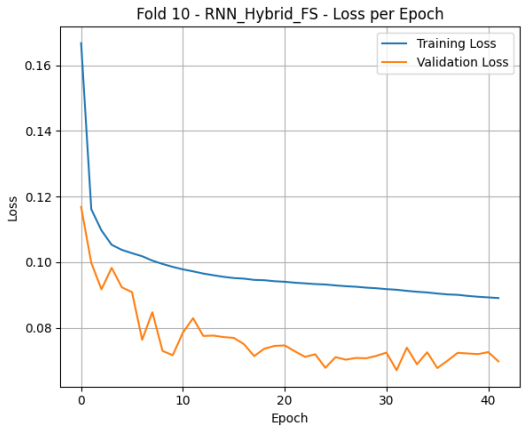

Fold 7 - RNN_Hybrid_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

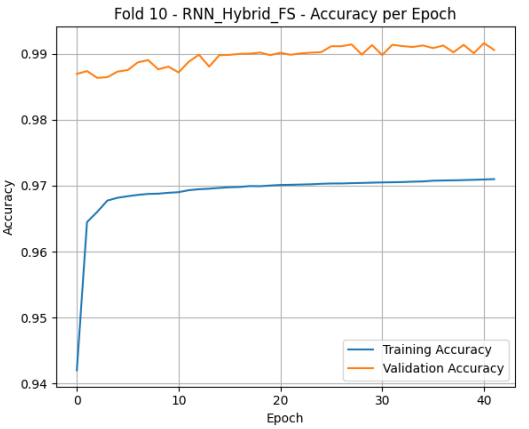

Fold 8 - RNN_Hybrid_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch


Fold 8 - RNN_Hybrid_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch


Fold 9 - RNN_Hybrid_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch


Fold 9 - RNN_Hybrid_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch


Fold 10 - RNN_Hybrid_FS - Loss per Epoch
Training Loss
Validation Loss
Loss
Epoch


Fold 10 - RNN_Hybrid_FS - Accuracy per Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch

**PREPARATION**

```
#library
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from imblearn.over_sampling import SMOTE
from collections import Counter
import numpy as np # Impor numpy untuk menangani potensi nilai tak terbatas
import os
import random
import math
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import LSTM, Dropout, BatchNormalization, Reshape # Import necessary layers
import tensorflow as tf # Import tensorflow

# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Load dataset
df = pd.read_csv('/content/drive/MyDrive/DDOS2019/DDOS2019_30%_ALL.csv')

# Bersihkan nama kolom (hapus spasi di depan/di belakang)
df.columns = df.columns.str.strip()

# Verifikasi perubahan
df.columns

# Tampilkan semua fitur
print("Fitur yang tersedia:")
df.info()

df.head()

# Grafik jumlah label (imbalanced check)
plt.figure(figsize=(20,6))
ax = sns.countplot(data=df, x='Label', hue='Label', dodge=False,
                   order=df['Label'].value_counts().index,
                   palette='Paired', legend=False)
plt.title('Distribution of data')
plt.ylabel("Data points per Class")
# Menambahkan label nilai pada setiap bar
for container in ax.containers:
    ax.bar_label(container, fmt='%d', fontsize=10) # fmt='%d' untuk jumlah bilangan bulat
plt.show()
```

**SELEKSI FITURE**

**#LEARNING VEKTOR QUANTIZIATION (LVQ)**

```
# LVQ Khusus yang Disederhanakan untuk Pemilihan Fitur
class SimpleLVQFeatureSelection:
    def __init__(self, n_prototypes_per_class, learning_rate, min_learning_rate, decay_rate, max_epochs, n_features_to_select, random_state=None):
        self.n_prototypes_per_class = n_prototypes_per_class
        self.learning_rate = learning_rate
        self.min_learning_rate = min_learning_rate
        self.decay_rate = decay_rate
        self.max_epochs = max_epochs
        self.n_features_to_select = n_features_to_select
        self.random_state = random_state
        self.prototypes = []
        self.prototype_labels = []
        self.feature_weights = None
        self.feature_indices = None # Menyimpan indeks dari fitur asli
        self.selected_feature_indices = None

    def initialize_prototypes(self, X, y):
        if self.random_state is not None:
            random.seed(self.random_state)
            np.random.seed(self.random_state)

        self.prototypes = []
        self.prototype_labels = []
        unique_classes = np.unique(y)
        n_features = X.shape[1]
        self.feature_indices = np.arange(n_features)
        # Inisialisasi bobot fitur secara acak antara 0 dan 1
        self.feature_weights = np.random.rand(n_features)
        # Menormalkan bobot awal
        self.feature_weights /= np.sum(self.feature_weights)

        for class_label in unique_classes:
            class_indices = np.where(y == class_label)[0]
            if len(class_indices) < self.n_prototypes_per_class:
                # Menangani kasus di mana ada lebih sedikit sampel daripada prototipe yang diminta
                selected_indices = np.random.choice(class_indices, size=len(class_indices), replace=False)
            else:
```

```
            selected_indices =
np.random.choice(class_indices,
size=self.n_prototypes_per_class, replace=False)

            for idx in selected_indices:
                self.prototypes.append(X[idx])
                self.prototype_labels.append(class_label)

        self.prototypes = np.array(self.prototypes)
        self.prototype_labels =
np.array(self.prototype_labels)

    def update_prototypes(self, x, y_true, lr):
        # Hitung jarak Euclidean tertimbang ke semua
prototipe
        weighted_diff = (x - self.prototypes) *
self.feature_weights
        distances = np.sum(weighted_diff**2, axis=1)
        bmu_index = np.argmin(distances)
        bmu_prototype = self.prototypes[bmu_index]
        bmu_label = self.prototype_labels[bmu_index]

        # Perbarui prototipe
        if bmu_label == y_true:
            self.prototypes[bmu_index] += lr * (x -
bmu_prototype)
        else:
            self.prototypes[bmu_index] -= lr * (x -
bmu_prototype)

        # Perbarui bobot fitur (aturan pembaruan bobot
yang disederhanakan)
        # Pindahkan bobot ke arah 1 untuk fitur yang
berkontribusi pada klasifikasi yang benar
        # Pindahkan bobot ke arah 0 untuk fitur yang
berkontribusi pada klasifikasi yang salah
        weight_update = lr * (x - bmu_prototype) * (x -
bmu_prototype)
        if bmu_label == y_true:
            self.feature_weights += weight_update
        else:
            self.feature_weights -= weight_update

        # Pastikan bobot tidak bernilai negatif
        self.feature_weights = np.maximum(0,
self.feature_weights)
        # Menormalkan kembali bobot
        if np.sum(self.feature_weights) > 0:
            self.feature_weights /=
np.sum(self.feature_weights)

    def train(self, X, y):
        self.initialize_prototypes(X, y)
        lr = self.learning_rate
        epoch_accuracies = []
        epoch_feature_weights_sum = []

        for epoch in range(self.max_epochs):
            if lr < self.min_learning_rate:
                break

            # Mengacak data untuk setiap zaman
            if self.random_state is not None:
                indices = np.random.permutation(len(X))
                X_shuffled = X[indices]
                y_shuffled = y[indices]
            else:
                indices = np.random.permutation(len(X))
                X_shuffled = X[indices]
                y_shuffled = y[indices]

            for i in range(len(X_shuffled)):
                self.update_prototypes(X_shuffled[i],
y_shuffled[i], lr)

            # Laju pembelajaran peluruhan
            lr *= self.decay_rate

            #Menghitung dan menyimpan akurasi untuk
visualisasi
            y_pred = self.predict(X)
            accuracy = accuracy_score(y, y_pred)
            epoch_accuracies.append(accuracy)

epoch_feature_weights_sum.append(np.sum(self.fea
ture_weights)) # Hanya untuk memantau perubahan
bobot

            if (epoch + 1) % 10 == 0:
                print(f"Epoch {epoch +
1}/{self.max_epochs}, Learning Rate: {lr:.6f},
Accuracy: {accuracy:.4f}, Feature Weights Sum:
{np.sum(self.feature_weights):.4f}")

        # Pilih fitur berdasarkan bobot akhir
        sorted_feature_indices =
np.argsort(self.feature_weights)[::-1] # Sort
descending
        self.selected_feature_indices =
sorted_feature_indices[:self.n_features_to_select]

    def predict(self, X):
        predictions = []
        for x in X:
            # Hitung jarak tertimbang ke prototipe
            weighted_diff = (x - self.prototypes) *
self.feature_weights
            distances = np.sum(weighted_diff**2, axis=1)
            bmu_index = np.argmin(distances)

predictions.append(self.prototype_labels[bmu_index
])
        return np.array(predictions)
```

```
    def get_selected_features(self, feature_names):
        if self.selected_feature_indices is None:
            raise RuntimeError("Model must be trained
first to select features.")
        return [feature_names[i] for i in
self.selected_feature_indices]

    def get_feature_weights(self):
        return self.feature_weights

# --- LVQ Feature Selection Parameters ---
initial_epoch = 0 # Implementasi kami dimulai dari
epoch 0
max_epoch = 100
learning_rate = 0.00001
min_learning_rate = 0.000001
decay_rate = 0.95
prototypes_per_class = 5
features_to_select = 16
random_seed = 42 # Untuk kemampuan reproduksi

# Pisahkan fitur (X) dan target (y)
X = df.drop('Label', axis=1).values
y = df['Label'].values
feature_names = df.drop('Label',
axis=1).columns.tolist()

print(f"\nStarting LVQ Feature Selection with
{features_to_select} features...")

# Inisialisasi dan latih model pemilihan fitur LVQ
lvq_fs = SimpleLVQFeatureSelection(
    n_prototypes_per_class=prototypes_per_class,
    learning_rate=learning_rate,
    min_learning_rate=min_learning_rate,
    decay_rate=decay_rate,
    max_epochs=max_epoch,
    n_features_to_select=features_to_select,
    random_state=random_seed
)

lvq_fs.train(X, y)

# Dapatkan bobot fitur akhir SETELAH pelatihan
final_feature_weights = lvq_fs.get_feature_weights()

# Dapatkan fitur yang dipilih
selected_features =
lvq_fs.get_selected_features(feature_names)
selected_feature_indices =
lvq_fs.selected_feature_indices

print(f"\nSelected Features ({features_to_select}):")
# Sekarang final_feature_weights sudah ditentukan
for i, feature in enumerate(selected_features):
    # Dapatkan indeks asli dari fitur yang dipilih
    original_index = feature_names.index(feature)
    # Gunakan indeks asli untuk mendapatkan bobot
dari  final_feature_weights
    importance_score =
final_feature_weights[original_index]
    print(f"{i+1}. {feature}(Importance Score:
{importance_score:.4f})")

# --- Visualization ---
# visualisasikan Bobot Fitur
plt.figure(figsize=(15, 7))
# Urutkan fitur berdasarkan berat untuk visualisasi
yang lebih baik
sorted_indices =
np.argsort(final_feature_weights)[::-1]
sorted_weights =
final_feature_weights[sorted_indices]
sorted_feature_names = [feature_names[i] for i in
sorted_indices]

plt.bar(range(len(sorted_weights)), sorted_weights)
plt.xticks(range(len(sorted_weights)),
sorted_feature_names, rotation=90)
plt.xlabel('Features')
plt.ylabel('LVQ Feature Weight')
plt.title('LVQ Feature Weights After Training (Sorted)')
plt.tight_layout()
plt.show()

# Menyorot Fitur yang Dipilih dalam Plot Bobot
plt.figure(figsize=(15, 7))
colors = ['skyblue' if i in selected_feature_indices else
'salmon' for i in range(len(feature_names))]
plt.bar(range(len(feature_names)),
final_feature_weights, color=colors)
plt.xticks(range(len(feature_names)), feature_names,
rotation=90)
plt.xlabel('Features')
plt.ylabel('LVQ Feature Weight')
plt.title(f'LVQ Feature Weights and Selected Features
({features_to_select} features)')
plt.legend(['Selected', 'Not Selected'],
handles=[plt.Line2D([], [], color='skyblue', marker='s',
linestyle='None'),
plt.Line2D([], [], color='salmon', marker='s',
linestyle='None')])
plt.tight_layout()
plt.show()

# visualisasikan distribusi fitur yang dipilih vs fitur
yang tidak dipilih
if features_to_select > 0:
    print("\nMemvisualisasikan distribusi beberapa
fitur yang dipilih vs fitur yang tidak dipilih:")

    # Dapatkan nama beberapa fitur yang dipilih dan
yang tidak dipilih
```

```
    non_selected_feature_indices = [i for i in
range(len(feature_names)) if i not in
selected_feature_indices]

    features_to_compare = []
    if len(selected_feature_indices) > 0:
        features_to_compare.extend([feature_names[i]
for i in selected_feature_indices[:min(3,
len(selected_feature_indices))]])
    if len(non_selected_feature_indices) > 0:
        features_to_compare.extend([feature_names[i]
for i in non_selected_feature_indices[:min(3,
len(non_selected_feature_indices))]])

    if features_to_compare:
        n_cols = 3
        n_rows = (len(features_to_compare) + n_cols -
1) // n_cols
        plt.figure(figsize=(n_cols * 6, n_rows * 5))

        for i, feature in
enumerate(features_to_compare):
            plt.subplot(n_rows, n_cols, i + 1)
            sns.boxplot(data=df, x='Label', y=feature)
            plt.title(f'Distribution of {feature} by Label')
            plt.xlabel('Label (0: BENIGN, 1: Attack)')
            plt.ylabel(feature)

        plt.tight_layout()
        plt.show()
    else:
        print("Not enough features to compare selected
vs non-selected distributions.")
else:
    print("\nNo features selected. Cannot visualize
distributions.")

# Sekarang Anda dapat menggunakan
df[selected_features] untuk langkah pemodelan
selanjutnya.
df_selected = df[selected_features].copy()
df_selected['Label'] = df['Label'] # Tambahkan
kembali kolom target

print("\nDataFrame with selected features:")
display(df_selected.head())
print(f"Shape of DataFrame with selected features:
{df_selected.shape}")

# Simpan DataFrame hasil seleksi fitur LVQ ke Google
Drive
output_path_fs =
'/content/drive/MyDrive/DDOS2019/normalisai_trai
n/Dataset2019_FS_LVQ_30%.csv'
df_selected.to_csv(output_path_fs, index=False)
print(f"\nDataFrame hasil seleksi fitur LVQ disimpan
ke: {output_path_fs}")
```

**#AUTOENCODER**

```
# --- Tentukan X dan y (digunakan kembali dari sel
LVQ) ---
# Pisahkan fitur (X) dan target (y)
X = df.drop('Label', axis=1).values
y = df['Label'].values
feature_names = df.drop('Label',
axis=1).columns.tolist()
# --- Akhir dari kode yang digunakan kembali ---

# --- Autoencoder Parameters ---
# Bentuk input akan ditentukan oleh jumlah fitur
dalam X sebelum pemisahan
input_dim = X.shape[1] # Mengatur input_dim secara
dinamis berdasarkan X
output_dim = input_dim
hidden_dim1 = 64
hidden_dim2 = 32
hidden_dim3 = 16 # Ini adalah lapisan bottleneck
untuk pengurangan fitur
learning_rate_ae = 0.00001
epochs_ae = 100
batch_size_ae = 256
# Fungsi kerugian dan pengoptimal ditentukan dalam
deskripsi masalah

# --- Persiapan Data untuk Autoencoder ---
# Periksa bentuk X untuk mengonfirmasi input_dim
print(f"\nShape of X for Autoencoder training:
{X.shape}")
print(f"Expected input dimension: {input_dim}")

# --- Membangun Model Autoencoder ---

# Encoder
input_layer = Input(shape=(input_dim,))
encoder_h1 = Dense(hidden_dim1,
activation='relu')(input_layer)
encoder_h2 = Dense(hidden_dim2,
activation='relu')(encoder_h1)
# Bottleneck layer - ini akan menjadi representasi
terkompresi kami
bottleneck = Dense(hidden_dim3,
activation='relu')(encoder_h2)

# Decoder
decoder_h3 = Dense(hidden_dim2,
activation='relu')(bottleneck)
decoder_h2 = Dense(hidden_dim1,
activation='relu')(decoder_h3)
# gunakan Sigmoid untuk tugas rekonstruksi karena
sesuai dengan rentang data [0, 1].
output_layer = Dense(output_dim,
activation='sigmoid')(decoder_h2)

# Model enkoder otomatis (enkoder + dekoder)
```

```
autoencoder = Model(inputs=input_layer, outputs=output_layer)

# Model encoder (untuk mendapatkan fitur yang dikompresi)
encoder_model = Model(inputs=input_layer, outputs=bottleneck)

# --- Mengkompilasi Autoencoder---
optimizer_ae = Adam(learning_rate=learning_rate_ae)
loss_function_ae = 'binary_crossentropy' # Or 'mse'

print(f"\nCompiling Autoencoder with Loss: {loss_function_ae}, Optimizer: Adam(lr={learning_rate_ae})")
autoencoder.compile(optimizer=optimizer_ae, loss=loss_function_ae)

# --- Train the Autoencoder ---
print(f"\nTraining Autoencoder for {epochs_ae} epochs...")
history = autoencoder.fit(X, X, # Latih untuk merekonstruksi input X
                          epochs=epochs_ae,
                          batch_size=batch_size_ae,
                          shuffle=True, # Mengacak data untuk setiap zaman
                          validation_split=0.1, # Gunakan set validasi yang kecil
                          verbose=1) # Tampilkan kemajuan

print("\nAutoencoder training finished.")

# --- Feature Selection using Autoencoder ---
X_compressed = encoder_model.predict(X)

print(f"\nShape of compressed features from autoencoder bottleneck: {X_compressed.shape}")
print(f"Number of features after compression: {X_compressed.shape[1]}")

# melihat bobot dari lapisan input ke lapisan tersembunyi pertama, Menjumlahkan bobot absolut untuk setiap fitur input di semua node di H1. Fitur dengan jumlah yang lebih tinggi berpotensi lebih penting.
# Dapatkan bobot dari lapisan padat pertama (Masukan ke Lapisan Tersembunyi 1)
input_to_h1_weights = autoencoder.layers[1].get_weights()[0] # [0] untuk bobot, [1] untuk bias

# Hitung nilai kepentingan untuk setiap fitur asli
feature_importance_scores = np.sum(np.abs(input_to_h1_weights), axis=1)

# Dapatkan nama fitur asli (tidak termasuk 'Label')
original_feature_names = df.drop('Label', axis=1).columns.tolist()

# Urutkan fitur berdasarkan skor kepentingan dalam urutan menurun
sorted_feature_indices_ae = np.argsort(feature_importance_scores)[::-1]
sorted_feature_scores_ae = feature_importance_scores[sorted_feature_indices_ae]
sorted_feature_names_ae = [original_feature_names[i] for i in sorted_feature_indices_ae]

# Pilih fitur 'hidden_dim3'=bottleneck (16) teratas berdasarkan skor kepentingan ini
num_features_to_select_ae = hidden_dim3 # Pilih 16 fitur asli

selected_feature_indices_ae = sorted_feature_indices_ae[:num_features_to_select_ae]
selected_feature_names_ae = sorted_feature_names_ae[:num_features_to_select_ae]

print(f"\nSelected Original Features based on Autoencoder Input Layer Weights ({num_features_to_select_ae}):")
for i, feature in enumerate(selected_feature_names_ae):
    print(f"{i+1}. {feature} (Importance Score: {sorted_feature_scores_ae[i]:.4f})")

# --- Visualization ---

# Visualisasikan Kerugian Pelatihan Autoencoder
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Training Loss')
if 'val_loss' in history.history:
    plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Autoencoder Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()

# Visualisasikan Skor Kepentingan Fitur Asli dari Bobot Autoencoder
plt.figure(figsize=(15, 7))
plt.bar(range(len(sorted_feature_scores_ae)), sorted_feature_scores_ae)
plt.xticks(range(len(sorted_feature_scores_ae)), sorted_feature_names_ae, rotation=90)
```

```
plt.xlabel('Original Features')
plt.ylabel('Importance Score (Sum of Abs Weights)')
plt.title('Original Feature Importance Scores based
on Autoencoder Input Layer Weights')
plt.tight_layout()
plt.show()

# Menyoroti Fitur yang Dipilih dalam Plot Penting
plt.figure(figsize=(15, 7))
colors_ae = ['skyblue' if i in
selected_feature_indices_ae else 'salmon' for i in
range(len(original_feature_names))]
plt.bar(range(len(original_feature_names)),
feature_importance_scores, color=colors_ae)
plt.xticks(range(len(original_feature_names)),
original_feature_names, rotation=90)
plt.xlabel('Original Features')
plt.ylabel('Importance Score (Sum of Abs Weights)')
plt.title(f'Original Feature Importance and Selected
Features ({num_features_to_select_ae} features)
from Autoencoder')
plt.legend(['Selected', 'Not Selected'],
handles=[plt.Line2D([], [], color='skyblue', marker='s',
linestyle='None'),
plt.Line2D([], [], color='salmon', marker='s',
linestyle='None')])
plt.tight_layout()
plt.show()

# membuat DataFrame baru hanya dengan fitur-fitur
yang dipilih ini dan label target.
df_selected_autoencoder_original =
df[selected_feature_names_ae].copy()
df_selected_autoencoder_original['Label'] =
df['Label'] # Tambahkan kolom target kembali

print("\nDataFrame with selected ORIGINAL features
based on Autoencoder:")
print(df_selected_autoencoder_original.head())
print(f"Shape of DataFrame with selected ORIGINAL
features:
{df_selected_autoencoder_original.shape}")

# Simpan DataFrame hasil seleksi fitur Autoencoder
ke Google Drive
output_path_fs_ae =
'/content/drive/MyDrive/DDOS2019/normalisai_trai
n/Dataset2019_FS_Autoencoder_30%.csv'
df_selected_autoencoder_original.to_csv(output_pa
th_fs_ae, index=False)
print(f"\nDataFrame hasil seleksi fitur Autoencoder
(Original Features) disimpan ke:
{output_path_fs_ae}")

#SELEKSI FITUR SECARA GABUNGAN
# --- Hybrid Feature Selection (LVQ + Autoencoder) ---
print("\n--- Starting Hybrid Feature Selection (LVQ +
Autoencoder) ---")

lvq_norm_scores = final_feature_weights

# Menormalkan skor kepentingan Autoencoder
(penskalaan Min-Maks)
# Hindari pembagian dengan nol jika semua skor
sama
max_ae_score = np.max(feature_importance_scores)
min_ae_score = np.min(feature_importance_scores)

if max_ae_score - min_ae_score > 0:
  ae_norm_scores = (feature_importance_scores -
min_ae_score) / (max_ae_score - min_ae_score)
else:
  # Jika semua skor sama, tetapkan skor yang
dinormalisasi ke nilai yang seragam (misalnya, 0,5
atau 1/fitur)
  print("Warning: All Autoencoder feature
importance scores are the same.")
  ae_norm_scores =
np.ones_like(feature_importance_scores) /
len(feature_importance_scores)

# Menggabungkan nilai (misalnya, jumlah atau rata-
rata sederhana)
# Penjumlahan adalah cara yang mudah untuk
menggabungkan
combined_scores = lvq_norm_scores +
ae_norm_scores # Penjumlahan sederhana

# Urutkan fitur berdasarkan skor gabungan
sorted_combined_indices =
np.argsort(combined_scores)[::-1] # Sort descending
sorted_combined_scores =
combined_scores[sorted_combined_indices]
sorted_combined_feature_names =
[original_feature_names[i] for i in
sorted_combined_indices]

# --- Pilih N fitur teratas dari peringkat gabungan ---
selected_feature_indices_hybrid =
sorted_combined_indices[:num_features_to_select_
hybrid]
selected_feature_names_hybrid =
sorted_combined_feature_names[:num_features_to
_select_hybrid]

print(f"\nSelected ORIGINAL Features based on
HYBRID LVQ + Autoencoder Scores
({num_features_to_select_hybrid}):")
for i, feature in
enumerate(selected_feature_names_hybrid):
  print(f"{i+1}. {feature} (Combined Score:
{sorted_combined_scores[i]:.4f})")
```

```
# --- Visualisasi Skor Gabungan ---

plt.figure(figsize=(15, 7))
plt.bar(range(len(sorted_combined_scores)),
sorted_combined_scores)
plt.xticks(range(len(sorted_combined_scores)),
sorted_combined_feature_names, rotation=90)
plt.xlabel('Original Features')
plt.ylabel('Combined Importance Score (Normalized
LVQ + Normalized AE)')
plt.title('Hybrid Feature Importance Scores (LVQ +
Autoencoder)')
plt.tight_layout()
plt.show()

# Sorot Fitur yang Dipilih dalam Plot Skor Gabungan
plt.figure(figsize=(15, 7))
colors_hybrid = ['skyblue' if i in
selected_feature_indices_hybrid else 'salmon' for i in
range(len(original_feature_names))]
plt.bar(range(len(original_feature_names)),
combined_scores, color=colors_hybrid)
plt.xticks(range(len(original_feature_names)),
original_feature_names, rotation=90)
plt.xlabel('Original Features')
plt.ylabel('Combined Importance Score')
plt.title(f'Hybrid Feature Importance and Selected
Features ({num_features_to_select_hybrid} features)
from LVQ + Autoencoder')
plt.legend(['Selected', 'Not Selected'],
handles=[plt.Line2D([], [], color='skyblue', marker='s',
linestyle='None'),
plt.Line2D([], [], color='salmon', marker='s',
linestyle='None')])
plt.tight_layout()
plt.show()

# Buat DataFrame baru hanya dengan fitur-fitur yang
dipilih dan label target.
df_selected_hybrid =
df[selected_feature_names_hybrid].copy()
df_selected_hybrid['Label'] = df['Label'] # Tambahkan
kolom target kembali

print("\nDataFrame with selected ORIGINAL features
based on HYBRID LVQ + Autoencoder:")
print(df_selected_hybrid.head())
print(f"Shape of DataFrame with selected ORIGINAL
features (Hybrid): {df_selected_hybrid.shape}")

# --- Simpan Fitur Hibrida yang Dipilih DataFrame ---
output_path_fs_hybrid =
'/content/drive/MyDrive/DDOS2019/normalisai_trai
n/Dataset2019_FS_Hybrid_30%.csv'
df_selected_hybrid.to_csv(output_path_fs_hybrid,
index=False)
print(f"\nDataFrame hasil seleksi fitur Hybrid (LVQ +
Autoencoder) disimpan ke:
{output_path_fs_hybrid}")
print("\n--- Hybrid Feature Selection Completed ---")

#TAMPILKAN NILAI SECARA KESELURUHAN UNTUK
MASING-MASING FITURE SELEKTION
# Membuat DataFrame untuk menampilkan skor
kepentingan fitur
feature_importance_data = {
  'Feature': sorted_feature_names, # Features
sorted by LVQ weight
  'LVQ Importance Score': sorted_weights, # LVQ
weights
  'AE Importance Score':
sorted_feature_scores_ae[np.argsort(sorted_feature
_indices_ae)] # Skor AE diselaraskan dengan urutan
pengurutan LVQ (atau pengurutan secara
independen)
  # Untuk menyelaraskan skor AE ke urutan yang
sama dengan sorted_feature_names (urutan LVQ),
kita memerlukan pemetaan
}

# Menyelaraskan kembali skor AE agar sesuai dengan
urutan nama fitur yang diurutkan LVQ
ae_scores_dict = dict(zip(original_feature_names,
feature_importance_scores))
ae_scores_aligned_to_lvq_order =
[ae_scores_dict.get(feature, 0) for feature in
sorted_feature_names]
feature_importance_data['AE Importance Score'] =
ae_scores_aligned_to_lvq_order

# Tambahkan skor gabungan, selaras dengan urutan
yang sama
combined_scores_dict =
dict(zip(original_feature_names, combined_scores))
combined_scores_aligned_to_lvq_order =
[combined_scores_dict.get(feature, 0) for feature in
sorted_feature_names]

feature_importance_data['Hybrid Importance Score']
= combined_scores_aligned_to_lvq_order

df_feature_importance =
pd.DataFrame(feature_importance_data)

# Urutkan tabel akhir berdasarkan salah satu skor
kepentingan (misalnya, Hibrida)
df_feature_importance =
df_feature_importance.sort_values(by='Hybrid
Importance Score',
ascending=False).reset_index(drop=True)

print("\nFeature Importance Scores:")
# Display the DataFrame
```

```
from IPython.display import display
display(df_feature_importance)

# Simpan fitur penting DataFrame ke Google Drive
output_dir =
'/content/drive/MyDrive/DDOS2019/normalisai_trai
n/DDOS2019_30%'
output_path_importance = os.path.join(output_dir,
'Feature_Importance_Scores.csv')

# Buat direktori jika tidak ada (pastikan os diimpor di
sel sebelumnya atau di sini)
if not os.path.exists(output_dir):
  os.makedirs(output_dir)
  print(f"Created directory: {output_dir}")

df_feature_importance.to_csv(output_path_importa
nce, index=False)
print(f"\nFeature importance scores saved to:
{output_path_importance}")
```

**#MODELKAN RNN BERBASIS AUTOENCODER**

```
# --- Parameters for RNN-LSTM Model ---

if 'df_hybrid_selected' in locals() and not
df_hybrid_selected.empty:
  n_features_rnn = df_hybrid_selected.shape[1] - 1
# Kurangi kolom 'Label'
else:
  # Fallback: gunakan jumlah fitur dari df asli jika
pemilihan hibrida gagal atau tidak dijalankan
  print("Warning: 'df_hybrid_selected' not found or
is empty. Using original features for RNN input
dimension.")
  n_features_rnn = df.shape[1] - 1 # Subtract the
'Label' column
  df_hybrid_selected = df.copy() # Gunakan bingkai
data lengkap sebagai fallback

# Membentuk ulang data untuk LSTM: (sampel,
langkah waktu, fitur)
X_rnn = df_hybrid_selected.drop('Label',
axis=1).values
y_rnn = df_hybrid_selected['Label'].values

# Ubah bentuk X_rnn agar memiliki dimensi waktu.
Setiap sampel adalah 1 timestep.
X_rnn = X_rnn.reshape((X_rnn.shape[0], 1,
X_rnn.shape[1])) # (samples, 1, n_features_rnn)

print(f"\nShape of data for RNN: {X_rnn.shape}")
print(f"Number of features per timestep:
{n_features_rnn}")

# Tentukan parameter model sesuai kebutuhan
lstm_units_1 = 64
dropout_rate = 0.2
lstm_units_2 = 32
dense_units = 16
output_units = 1 # For binary classification

# Activation functions
dense_activation = 'relu'
output_activation = 'sigmoid' # For binary
classification

# Optimizer
optimizer_rnn = Adam() # Menggunakan laju
pembelajaran default pada awalnya, dapat
menentukan jika diperlukan:
Adam(learning_rate=0.001)

# Loss Function
loss_function_rnn = 'binary_crossentropy' # Untuk
klasifikasi biner

# --- Membangun Arsitektur Model RNN-LSTM ---

# Lapisan masukan yang mengharapkan data bentuk
(langkah waktu, fitur)
rnn_input = Input(shape=(X_rnn.shape[1],
X_rnn.shape[2]))

# LSTM Layer 1
# return_sequences = True karena kita menumpuk
lapisan LSTM yang lain
lstm_h1 = LSTM(lstm_units_1,
return_sequences=True)(rnn_input)
# lstm_h1 = BatchNormalization()(lstm_h1)
lstm_h1 = Dropout(dropout_rate)(lstm_h1)

# LSTM Layer 2
# return_sequences = False untuk lapisan LSTM
terakhir sebelum lapisan Dense
lstm_h2 = LSTM(lstm_units_2,
return_sequences=False)(lstm_h1)
# lstm_h2 = BatchNormalization()(lstm_h2)
lstm_h2 = Dropout(dropout_rate)(lstm_h2)

# Dense Layer
dense_layer = Dense(dense_units,
activation=dense_activation)(lstm_h2)

# Output Layer
output_layer_rnn = Dense(output_units,
activation=output_activation)(dense_layer)

# Create the Model
rnn_model = Model(inputs=rnn_input,
outputs=output_layer_rnn)

# --- Compile the Model ---
rnn_model.compile(optimizer=optimizer_rnn,
          loss=loss_function_rnn,
```

```
            metrics=['accuracy']) # Include accuracy as
a metric

# --- Display Model Summary ---
print("\nRNN-LSTM Model Architecture:")
rnn_model.summary()
```

**TRAIN DAN VALIDASI**
**#LEARNING VECTOR QUANTIZIASION**

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import time
import warnings
from sklearn.metrics import confusion_matrix,
roc_curve, auc, ConfusionMatrixDisplay
from tensorflow.keras.callbacks import EarlyStopping
import os # Import os
from google.colab import drive # Import drive
import tensorflow as tf # Import tensorflow
from tensorflow.keras.models import Sequential #
Import Sequential
from tensorflow.keras.layers import LSTM, Dropout,
Dense # Import layers
from tensorflow.keras.optimizers import Adam #
Import Adam
from tensorflow.keras import Input # Import Input
from sklearn.model_selection import StratifiedKFold
# Import StratifiedKFold
from imblearn.over_sampling import SMOTE #
Import SMOTE
from collections import Counter # Import Counter
import random # Import random
import seaborn as sns # Import seaborn
import json # Import json
from sklearn.preprocessing import MinMaxScaler #
Import MinMaxScaler for Min-Max normalization

warnings.filterwarnings('ignore')

# Pastikan reproduktifitas
seed_value = 42
np.random.seed(seed_value)
random.seed(seed_value)
tf.random.set_seed(seed_value)

# Pasang Google Drive jika belum terpasang.
try:
  drive.mount('/content/drive',
force_remount=True)
  print("Google Drive mounted.")
except Exception as e:
  print(f"Error mounting Google Drive: {e}")

# Tentukan jalur file untuk dataset LVQ
file_path_lvq =
'/content/drive/MyDrive/DDOS2019/normalisai_trai
n/Dataset2019_FS_LVQ_30%.csv'

df_lvq = None
X_full_lvq = None
y_full_lvq = None

all_labels_lvq = None
all_labels_mapped_lvq = None
num_classes_lvq = 0

# Muat dataset LVQ dan periksa keberadaan file.
if os.path.exists(file_path_lvq):
  try:
    df_lvq = pd.read_csv(file_path_lvq)
    X_full_lvq = df_lvq.drop('Label', axis=1).values
    y_full_lvq = df_lvq['Label'].values
    print(f"Successfully loaded: {file_path_lvq}")
  except Exception as e:
    print(f"Error loading {file_path_lvq}: {e}")
else:
  print(f"File not found: {file_path_lvq}")

# Lanjutkan hanya jika dataset LVQ telah dimuat
dengan sukses.
if X_full_lvq is not None and y_full_lvq is not None:
  all_labels_lvq = np.unique(y_full_lvq)
  num_classes_lvq = len(all_labels_lvq)

  # Mapping label ke bilangan bulat 0 dan 1 untuk
klasifikasi biner
  label_mapping_lvq = {label: i for i, label in
enumerate(sorted(all_labels_lvq))}
  y_full_lvq_mapped =
np.array([label_mapping_lvq[label] for label in
y_full_lvq])

  all_labels_mapped_lvq =
np.unique(y_full_lvq_mapped)

  # Tentukan fungsi pembangun model RNN untuk
klasifikasi biner.
  def build_rnn_model_binary(input_shape):
    model = Sequential()
    model.add(Input(shape=input_shape))
    model.add(LSTM(units=64,
return_sequences=True))
    model.add(Dropout(0.2))
    model.add(LSTM(units=32,
return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(units=16, activation='relu'))
    model.add(Dense(units=1, activation='sigmoid'))
    optimizer = Adam()
```

```
    model.compile(optimizer=optimizer,
loss='binary_crossentropy', metrics=['accuracy'])
    return model

  # --- Cross-Validation Setup ---
  n_splits = 10
  skf_lvq = StratifiedKFold(n_splits=n_splits,
shuffle=True, random_state=42)

  print(f"\nStarting {n_splits}-fold cross-validation
for LVQ FS...")

  # --- Tentukan Jalur Penyimpanan untuk Hasil
Sementara---
  # Simpan ke Google Drive untuk penyimpanan
permanen - jalur khusus untuk skenario LVQ
  intermediate_results_path_lvq =
'/content/drive/MyDrive/DDOS2019/normalisai_trai
n/intermediate_cv_results_lvq_only.json'

  # --- Muat Hasil yang Sudah Ada jika Tersedia ---
  results_lvq = {
    'RNN_LVQ_FS': {'accuracy': [], 'loss': [],
'conf_matrix': [], 'roc_auc': [], 'fpr': [], 'tpr': [], 'time':
[], 'history': []}
  }
  start_fold_lvq = 0

  if os.path.exists(intermediate_results_path_lvq):
    print(f"Loading existing results from:
{intermediate_results_path_lvq}")
    try:
      with open(intermediate_results_path_lvq, 'r')
as f:
        loaded_results = json.load(f)

      scenario_name = 'RNN_LVQ_FS'
      if scenario_name in loaded_results:
        data = loaded_results[scenario_name]
        results_lvq[scenario_name]['accuracy'] =
data.get('accuracy', [])
        results_lvq[scenario_name]['loss'] =
data.get('loss', [])
        # Konversikan daftar kembali menjadi array
numpy untuk matriks kebingungan.
        results_lvq[scenario_name]['conf_matrix'] =
[np.array(cm) for cm in data.get('conf_matrix', [])]
        results_lvq[scenario_name]['roc_auc'] =
data.get('roc_auc', [])
        # Konversikan daftar kembali menjadi array
numpy untuk fpr dan tpr
        results_lvq[scenario_name]['fpr'] =
[np.array(f) if f is not None else None for f in
data.get('fpr', [])]
        results_lvq[scenario_name]['tpr'] =
[np.array(t) if t is not None else None for t in
data.get('tpr', [])]
        results_lvq[scenario_name]['time'] =
data.get('time', [])
        results_lvq[scenario_name]['history'] =
data.get('history', [])

      # Tentukan lipatan awal
      completed_folds =
len(results_lvq[scenario_name]['accuracy'])
      start_fold_lvq = completed_folds
      print(f"Resuming from Fold {start_fold_lvq +
1}/{n_splits}")

    except Exception as e:
      print(f"Error loading results from
{intermediate_results_path_lvq}: {e}")
      print("Starting from the beginning.")
      # Reset hasil jika proses pemuatan gagal
      results_lvq = {
        'RNN_LVQ_FS': {'accuracy': [], 'loss': [],
'conf_matrix': [], 'roc_auc': [], 'fpr': [], 'tpr': [], 'time':
[], 'history': []}
      }
      start_fold_lvq = 0
  else:
    print("No existing results found. Starting from
the beginning.")


  # Parameter untuk Pelatihan RNN-LSTM
  epochs_cv = 50 # Mengubah epoch menjadi 50
  batch_size_cv = 64
  early_stopping_patience = 10  # menghentikan
proses secara dini

  # Tentukan callback penghentian dini
  early_stopping = EarlyStopping(monitor='val_loss',
patience=early_stopping_patience,
restore_best_weights=True, verbose=1)



  # --- ulangi Cross-Validation dari LVQ FS ---
  # Lalui setiap pembagian dan lewati lipatan yang
sudah selesai.
  splits_lvq = list(skf_lvq.split(X_full_lvq,
y_full_lvq_mapped))

  for fold_idx in range(start_fold_lvq, n_splits):
    fold = fold_idx + 1 # Sesuaikan jumlah lipatan
menjadi berbasis 1.
    train_index_lvq, test_index_lvq =
splits_lvq[fold_idx] # Dapatkan indeks untuk lipatan
saat ini

    print(f"\n--- Fold {fold}/{n_splits} ---")
```

```
        X_train_lvq, X_test_lvq =
X_full_lvq[train_index_lvq],
X_full_lvq[test_index_lvq]
        y_train_lvq, y_test_lvq =
y_full_lvq_mapped[train_index_lvq],
y_full_lvq_mapped[test_index_lvq]

        print(f"Train set shape (LVQ FS):
{X_train_lvq.shape}, Test set shape (LVQ FS):
{X_test_lvq.shape}")
        print(f"Train label distribution:
{Counter(y_train_lvq)}")
        print(f"Test label distribution:
{Counter(y_test_lvq)}")

        # Inisialisasi metrik untuk lipatan ini ke nilai
default/NaN.
        accuracy = np.nan
        loss = np.nan
        cm = np.zeros((num_classes_lvq,
num_classes_lvq), dtype=int) # Inisialisasi matriks
kebingungan menjadi nol
        roc_auc_lvq = np.nan
        fpr_lvq = None
        tpr_lvq = None
        train_test_time = np.nan
        history_lvq = None

        # --- Normalisasi data pelatihan sebelum SMOTE
menggunakan MinMaxScaler ---
        print("Menormalisasi data pelatihan sebelum
SMOTE menggunakan MinMaxScaler(LVQ FS)...")
        scaler = MinMaxScaler()
        X_train_lvq_normalized =
scaler.fit_transform(X_train_lvq)
        # Normalize test data using the scaler fitted on
training data
        X_test_lvq_normalized =
scaler.transform(X_test_lvq)
        print("Normalization  selesai.")

        # --- Terapkan SMOTE pada data pelatihan yang
dinormalisasi untuk lipatan saat ini. ---
        print("Menerapkan SMOTE pada data pelatihan
yang dinormalisasi (LVQ FS)...")
        smote = SMOTE(random_state=42 + fold_idx) #
Gunakan fold_idx untuk konsistensi acak per
pembagian.
        try:
            X_train_lvq_smote, y_train_lvq_smote =
smote.fit_resample(X_train_lvq_normalized,
y_train_lvq) # Terapkan SMOTE pada data yang telah
dinormalisasi
            print(f"Train set shape setelah SMOTE (LVQ
FS): {X_train_lvq_smote.shape}")
            print(f"Train label distribution setelah SMOTE:
{Counter(y_train_lvq_smote)}")

            # Mengubah bentuk data untuk RNN
(timesteps=1) - dipindahkan ke dalam blok try
            X_train_lvq_smote_rnn =
X_train_lvq_smote.reshape((X_train_lvq_smote.shap
e[0], 1, X_train_lvq_smote.shape[1]))
            X_test_lvq_rnn =
X_test_lvq_normalized.reshape((X_test_lvq_normali
zed.shape[0], 1, X_test_lvq_normalized.shape[1]))

            # --- Training RNN dengan LVQ FS ---
            print(f"Training RNN with LVQ FS (Fold
{fold})...")
            start_time = time.time()
            tf.keras.backend.clear_session()
            rnn_lvq_model =
build_rnn_model_binary(input_shape=(X_train_lvq_
smote_rnn.shape[1],
X_train_lvq_smote_rnn.shape[2]))

            # Latih model dengan Early Stopping dan
tampilkan kemajuan epoch.
            history_lvq = rnn_lvq_model.fit(
                X_train_lvq_smote_rnn, y_train_lvq_smote,
                epochs=epochs_cv,
                batch_size=batch_size_cv,
                verbose=1, # tampilkan epoch progress
                validation_data=(X_test_lvq_rnn,
y_test_lvq),
                callbacks=[early_stopping]
            )

            end_time = time.time()
            train_test_time = end_time - start_time

            # Evaluasi model dan hitung metrik setelah
pelatihan berhasil.
            loss, accuracy =
rnn_lvq_model.evaluate(X_test_lvq_rnn, y_test_lvq,
verbose=0)
            y_pred_prob =
rnn_lvq_model.predict(X_test_lvq_rnn)

            # Hitung ROC dan AUC untuk klasifikasi biner
            if len(np.unique(y_test_lvq)) < 2:
                print(f"  Mengabaikan perhitungan ROC AUC
untuk lipatan {fold} - LVQ FS karena hanya ada satu
kelas dalam set data uji..")
                fpr_lvq, tpr_lvq, roc_auc_lvq = None, None,
np.nan
            else:
                fpr_lvq, tpr_lvq, _ = roc_curve(y_test_lvq,
y_pred_prob)
                roc_auc_lvq = auc(fpr_lvq, tpr_lvq)

            y_pred = (y_pred_prob > 0.5).astype("int32")
```

```
        cm = confusion_matrix(y_test_lvq, y_pred,
labels=all_labels_mapped_lvq)

      except ValueError as e:
        print(f"Could not apply SMOTE for Fold {fold}
(LVQ FS): {e}")
        print("Skipping training for this fold.")
        # Metrik tetap dalam keadaan terinisialisasi.
(NaN/zeros/None)


      scenario_name = 'RNN_LVQ_FS'

results_lvq[scenario_name]['accuracy'].append(accur
acy)
      results_lvq[scenario_name]['loss'].append(loss)

results_lvq[scenario_name]['conf_matrix'].append(c
m)

results_lvq[scenario_name]['roc_auc'].append(roc_a
uc_lvq)

results_lvq[scenario_name]['fpr'].append(fpr_lvq)

results_lvq[scenario_name]['tpr'].append(tpr_lvq)

results_lvq[scenario_name]['time'].append(train_tes
t_time)

results_lvq[scenario_name]['history'].append(history
_lvq.history if history_lvq is not None else None) #
Pastikan riwayat disimpan atau Tidak
      print(f"  Completed Fold {fold} in
{train_test_time:.2f} seconds. Test Accuracy:
{accuracy:.4f}, Test Loss: {loss:.4f}")

      # --- Simpan Hasil Sementara Setelah Setiap
Lipatan ---
      print(f"Saving intermediate results after Fold
{fold}...")
      try:
        # Konversi array numpy menjadi daftar untuk
serialisasi JSON, termasuk penanganan nilai None
dan NaN.
        results_to_save = {}
        scenario_name_save = 'RNN_LVQ_FS'
        data_to_save =
results_lvq[scenario_name_save]
        results_to_save[scenario_name_save] = {
          'accuracy': data_to_save['accuracy'],
          'loss': data_to_save['loss'],
          'conf_matrix': [cm_item.tolist() if cm_item is
not None else None for cm_item in
data_to_save['conf_matrix']], # Convert numpy array
to list, handle None
          'roc_auc': data_to_save['roc_auc'],
          'fpr': [f.tolist() if f is not None else None for f
in data_to_save['fpr']], # Convert numpy array to list,
handle None
          'tpr': [t.tolist() if t is not None else None for
t in data_to_save['tpr']], # Convert numpy array to
list, handle None
          'time': data_to_save['time'],
          'history': data_to_save['history'] # Sejarah
sudah ditentukan sebagai daftar atau None.
        }

        with open(intermediate_results_path_lvq, 'w')
as f:
          json.dump(results_to_save, f)
        print("Intermediate results saved.")
      except Exception as e_save:
        print(f"Error saving intermediate results:
{e_save}")
  print("\n--- Cross-Validation for LVQ FS Completed
---")

  # --- Mengumpulkan dan Melaporkan Hasil untuk
LVQ FS ---
  print("\n--- Aggregated Results for LVQ FS (Mean
+/- Std Dev) ---")

  scenario_name = 'RNN_LVQ_FS'
  result_data = results_lvq[scenario_name]

  # Hitung rata-rata dan simpangan baku untuk
setiap metrik di seluruh lipatan, dengan menangani
nilai NaN.
  avg_accuracy =
np.nanmean(result_data['accuracy']) if
result_data['accuracy'] else np.nan
  std_accuracy = np.nanstd(result_data['accuracy'])
if result_data['accuracy'] else np.nan
  avg_loss = np.nanmean(result_data['loss']) if
result_data['loss'] else np.nan
  std_loss = np.nanstd(result_data['loss']) if
result_data['loss'] else np.nan

  valid_roc_auc = [auc for auc in
result_data['roc_auc'] if auc is not None and not
np.isnan(auc)]
  avg_roc_auc = np.mean(valid_roc_auc) if
valid_roc_auc else np.nan
  std_roc_auc = np.std(valid_roc_auc) if
valid_roc_auc else np.nan

  valid_total_times = [t for t in result_data['time'] if t
is not None and not np.isnan(t)]
  avg_total_time = np.mean(valid_total_times) if
valid_total_times else np.nan
  std_total_time = np.std(valid_total_times) if
valid_total_times else np.nan
```

```
    print(f"\nScenario: {scenario_name}")
    print(f"  Average Test Accuracy: {avg_accuracy:.4f} +/- {std_accuracy:.4f}")
    print(f"  Average Test Loss: {avg_loss:.4f} +/- {std_loss:.4f}")
    print(f"  Average ROC AUC: {avg_roc_auc:.4f} +/- {std_roc_auc:.4f}" if not np.isnan(avg_roc_auc) else "  Average ROC AUC: N/A")
    print(f"  Average Total Training/Testing Time: {avg_total_time:.4f} +/- {std_total_time:.4f} seconds" if not np.isnan(avg_total_time) else "  Average Total Training/Testing Time: N/A")

    # --- Visualisasi dari LVQ FS ---

    # 1. Visualisasikan Matriks Kebingungan Rata-Rata
    print("\n--- Visualisasi Matriks Konfusi Total untuk LVQ FS ---")
    plt.figure(figsize=(6, 5))
    if len(result_data['conf_matrix']) > 0 and not all(np.all(cm == 0) for cm in result_data['conf_matrix'] if cm is not None):
        valid_cms = [cm for cm in result_data['conf_matrix'] if cm is not None and not np.all(cm == 0)]
        if valid_cms:
            sum_cm = np.sum(valid_cms, axis=0)
        else:
            sum_cm = np.zeros((num_classes_lvq, num_classes_lvq), dtype=int)

        all_labels_list_for_viz_lvq = [str(label) for label in sorted(list(all_labels_mapped_lvq))]
        disp = ConfusionMatrixDisplay(confusion_matrix=sum_cm, display_labels=all_labels_list_for_viz_lvq)
        disp.plot(cmap=plt.cm.Blues, values_format='d')
        plt.title(f'{scenario_name}\nTotal Confusion Matrix')
        plt.xlabel('Predicted Label')
        plt.ylabel('True Label')
        plt.tight_layout()
        plt.show()
    else:
        print(f"Tidak ada matriks kebingungan yang valid tersedia untuk{scenario_name}.")

    # 2. Visualize Average ROC Curve
    print("\n--- Visualizing Average ROC Curve for LVQ FS ---")
    plt.figure(figsize=(8, 7))
    valid_fprs = [f for f in result_data.get('fpr', []) if f is not None and not np.isnan(f).any()]
    valid_tprs = [t for t in result_data.get('tpr', []) if t is not None and not np.isnan(t).any()]

    if len(valid_tprs) > 0:
        mean_fpr = np.linspace(0, 1, 100)
        tprs_interp = []
        for i in range(len(valid_tprs)):
            tprs_interp.append(np.interp(mean_fpr, valid_fprs[i], valid_tprs[i]))
            tprs_interp[-1][0] = 0.0

        if len(tprs_interp) > 0:
            mean_tpr = np.mean(tprs_interp, axis=0)
            mean_tpr[-1] = 1.0
            avg_roc_auc = auc(mean_fpr, mean_tpr)

            plt.plot(mean_fpr, mean_tpr,
                     label=f'{scenario_name} (Avg AUC = {avg_roc_auc:.4f})',
                     lw=2, alpha=.8)
        else:
            print(f"  Not enough valid ROC curves to plot average for {scenario_name}.")
    else:
        print(f"  No valid ROC curves available for {scenario_name}.")

    plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random Guess', alpha=.8)
    plt.xlim([-0.05, 1.05])
    plt.ylim([-0.05, 1.05])
    plt.xlabel('False Positive Rate (FPR)')
    plt.ylabel('True Positive Rate (TPR)')
    plt.title('Average Receiver Operating Characteristic (ROC) Curve (LVQ FS)')
    plt.legend(loc="lower right")
    plt.grid(True)
    plt.show()

    # 3. Visualize Average Training and Validation Metrics
    print("\n--- Visualizing Average Training and Validation Metrics for LVQ FS ---")
    plt.figure(figsize=(14, 6))

    # Accuracy Plot
    plt.subplot(1, 2, 1)
    if len(result_data.get('history', [])) > 0:
        histories = [h for h in result_data['history'] if h is not None]
        if histories:
            min_epochs = min([len(h.get('accuracy', [])) for h in histories], default=0)
            if min_epochs > 0:
                train_accs = [h.get('accuracy', [])[:min_epochs] for h in histories if len(h.get('accuracy', [])) >= min_epochs]
                val_accs = [h.get('val_accuracy', [])[:min_epochs] for h in histories if len(h.get('val_accuracy', [])) >= min_epochs]
                if train_accs and val_accs:
```

```
            avg_train_acc = np.mean(train_accs,
axis=0)
            avg_val_acc = np.mean(val_accs, axis=0)
            plt.plot(range(min_epochs),
avg_train_acc, label=f'{scenario_name} Train Acc',
alpha=0.7)
            plt.plot(range(min_epochs), avg_val_acc,
label=f'{scenario_name} Val Acc', linestyle='--',
alpha=0.7)
          else:
            print(f"  Not enough consistent history
data for plotting average accuracy for
{scenario_name}.")
        else:
          print(f"  No valid history data found for
plotting average accuracy for {scenario_name}.")
      else:
        print(f"  No valid history found for
{scenario_name} after filtering None.")

  plt.title('Average Model Accuracy per Epoch (LVQ
FS)')
  plt.xlabel('Epoch')
  plt.ylabel('Accuracy')
  plt.legend()
  plt.grid(True)

  # Loss Plot
  plt.subplot(1, 2, 2)
  if len(result_data.get('history', [])) > 0:
    histories = [h for h in result_data['history'] if h is
not None]
    if histories:
      min_epochs = min([len(h.get('loss', [])) for h in
histories], default=0)
      if min_epochs > 0:
        train_losses = [h.get('loss', [])[:min_epochs]
for h in histories if len(h.get('loss', [])) >=
min_epochs]
        val_losses = [h.get('val_loss',
[])[:min_epochs] for h in histories if len(h.get('loss',
[])) >= min_epochs]
        if train_losses and val_losses:
          avg_train_loss = np.mean(train_losses,
axis=0)
          avg_val_loss = np.mean(val_losses,
axis=0)
          plt.plot(range(min_epochs),
avg_train_loss, label=f'{scenario_name} Train Loss',
alpha=0.7)
          plt.plot(range(min_epochs), avg_val_loss,
label=f'{scenario_name} Val Loss', linestyle='--',
alpha=0.7)
        else:
          print(f"  Not enough consistent history
data for plotting average loss for {scenario_name}.")
      else:
        print(f"  No valid history data found for
plotting average loss for {scenario_name}.")
    else:
      print(f"  No valid history found for
{scenario_name} after filtering None.")

  plt.title('Average Model Loss per Epoch (LVQ FS)')
  plt.xlabel('Epoch')
  plt.ylabel('Loss')
  plt.legend()
  plt.grid(True)

  plt.tight_layout()
  plt.show()

  print("\n--- Analysis and Visualization for LVQ FS
Completed ---")
```

**# AUTOENCODER**

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import time
import warnings
from sklearn.metrics import confusion_matrix,
roc_curve, auc, ConfusionMatrixDisplay
from tensorflow.keras.callbacks import EarlyStopping
import os # Import os
from google.colab import drive # Import drive
import tensorflow as tf # Import tensorflow
from tensorflow.keras.models import Sequential #
Import Sequential
from tensorflow.keras.layers import LSTM, Dropout,
Dense # Import layers
from tensorflow.keras.optimizers import Adam #
Import Adam
from tensorflow.keras import Input # Import Input
from sklearn.model_selection import StratifiedKFold
# Import StratifiedKFold
from imblearn.over_sampling import SMOTE #
Import SMOTE
from collections import Counter # Import Counter
import random # Import random
import seaborn as sns # Import seaborn
import json # Import json
from sklearn.preprocessing import MinMaxScaler #
Import MinMaxScaler for Min-Max normalization

warnings.filterwarnings('ignore')

# Pastikan reproduktifitas
seed_value = 42
np.random.seed(seed_value)
random.seed(seed_value)
tf.random.set_seed(seed_value)

# Pasang Google Drive jika belum terpasang.
```

```
try:
  drive.mount('/content/drive',
force_remount=True)
  print("Google Drive mounted.")
except Exception as e:
  print(f"Error mounting Google Drive: {e}")

# Tentukan jalur file untuk dataset Autoencoder
file_path_ae =
'/content/drive/MyDrive/DDOS2019/normalisai_trai
n/Dataset2019_FS_Autoencoder_30%.csv'

df_ae = None
X_full_ae = None
y_full_ae = None

all_labels_ae = None
all_labels_mapped_ae = None
num_classes_ae = 0

# Muat dataset Autoencoder dan periksa keberadaan
file.
if os.path.exists(file_path_ae):
  try:
    df_ae = pd.read_csv(file_path_ae)
    X_full_ae = df_ae.drop('Label', axis=1).values
    y_full_ae = df_ae['Label'].values
    print(f"Successfully loaded: {file_path_ae}")
  except Exception as e:
    print(f"Error loading {file_path_ae}: {e}")
else:
  print(f"File not found: {file_path_ae}")

# Lanjutkan hanya jika dataset Autoencoder telah
dimuat dengan sukses.
if X_full_ae is not None and y_full_ae is not None:
  all_labels_ae = np.unique(y_full_ae)
  num_classes_ae = len(all_labels_ae)

  # Mapping label ke bilangan bulat 0 dan 1 untuk
klasifikasi biner
  label_mapping_ae = {label: i for i, label in
enumerate(sorted(all_labels_ae))}
  y_full_ae_mapped =
np.array([label_mapping_ae[label] for label in
y_full_ae])

  all_labels_mapped_ae =
np.unique(y_full_ae_mapped)

  # Tentukan fungsi pembangun model RNN untuk
klasifikasi biner.
  def build_rnn_model_binary(input_shape):
    model = Sequential()
    model.add(Input(shape=input_shape))
    model.add(LSTM(units=64,
return_sequences=True))
    model.add(Dropout(0.2))
    model.add(LSTM(units=32,
return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(units=16, activation='relu'))
    model.add(Dense(units=1, activation='sigmoid'))
    optimizer = Adam()
    model.compile(optimizer=optimizer,
loss='binary_crossentropy', metrics=['accuracy'])
    return model

  # --- Cross-Validation Setup ---
  n_splits = 10
  skf_ae = StratifiedKFold(n_splits=n_splits,
shuffle=True, random_state=42)

  print(f"\nStarting {n_splits}-fold cross-validation
for Autoencoder FS...")

  # --- Tentukan jalur penyimpanan untuk hasil
sementara. ---
  # Simpan ke Google Drive untuk penyimpanan
permanen - jalur khusus untuk skenario Autoencoder
  intermediate_results_path_ae =
'/content/drive/MyDrive/DDOS2019/normalisai_trai
n/intermediate_cv_results_ae_only.json'

  # --- Muat Hasil yang Sudah Ada jika Tersedia ---
  results_ae = {
    'RNN_AE_FS': {'accuracy': [], 'loss': [],
'conf_matrix': [], 'roc_auc': [], 'fpr': [], 'tpr': [], 'time':
[], 'history': []}
  }
  start_fold_ae = 0

  if os.path.exists(intermediate_results_path_ae):
    print(f"Loading existing results from:
{intermediate_results_path_ae}")
    try:
      with open(intermediate_results_path_ae, 'r')
as f:
        loaded_results = json.load(f)

      scenario_name = 'RNN_AE_FS' # Satu-satunya
skenario dalam blok kode ini
      if scenario_name in loaded_results:
        data = loaded_results[scenario_name]
        results_ae[scenario_name]['accuracy'] =
data.get('accuracy', [])
        results_ae[scenario_name]['loss'] =
data.get('loss', [])
        # Konversikan daftar daftar kembali menjadi
array numpy untuk matriks kebingungan.
        results_ae[scenario_name]['conf_matrix'] =
[np.array(cm) for cm in data.get('conf_matrix', [])]
        results_ae[scenario_name]['roc_auc'] =
data.get('roc_auc', [])
```

```
            # Konversikan daftar daftar kembali menjadi
array numpy untuk fpr dan tpr
            results_ae[scenario_name]['fpr'] =
[np.array(f) if f is not None else None for f in
data.get('fpr', [])]
            results_ae[scenario_name]['tpr'] =
[np.array(t) if t is not None else None for t in
data.get('tpr', [])]
            results_ae[scenario_name]['time'] =
data.get('time', [])
            results_ae[scenario_name]['history'] =
data.get('history', [])

          # Tentukan lipatan awal
          completed_folds =
len(results_ae[scenario_name]['accuracy'])
          start_fold_ae = completed_folds
          print(f"Resuming from Fold {start_fold_ae +
1}/{n_splits}")

        except Exception as e:
          print(f"Error loading results from
{intermediate_results_path_ae}: {e}")
          print("Starting from the beginning.")
          # Reset hasil jika proses pemuatan gagal
          results_ae = {
            'RNN_AE_FS': {'accuracy': [], 'loss': [],
'conf_matrix': [], 'roc_auc': [], 'fpr': [], 'tpr': [], 'time':
[], 'history': []}
          }
          start_fold_ae = 0
    else:
      print("Tidak ditemukan hasil yang ada. Mulai
dari awal.")

    # Parameter untuk Pelatihan RNN-LSTM
    epochs_cv = 50
    batch_size_cv = 64
    early_stopping_patience = 10

    # Tentukan Early Stopping callback
    early_stopping = EarlyStopping(monitor='val_loss',
patience=early_stopping_patience,
restore_best_weights=True, verbose=1)

    # --- Cross-Validation Loop for Autoencoder FS ---
    # Lalui setiap pembagian dan lewati lipatan yang
sudah selesai.
    splits_ae = list(skf_ae.split(X_full_ae,
y_full_ae_mapped))

    for fold_idx in range(start_fold_ae, n_splits):
      fold = fold_idx + 1 # Sesuaikan jumlah lipatan
menjadi berbasis 1.
      train_index_ae, test_index_ae =
splits_ae[fold_idx] # Dapatkan indeks untuk lipatan
saat ini
      print(f"\n--- Fold {fold}/{n_splits} ---")

      X_train_ae, X_test_ae =
X_full_ae[train_index_ae], X_full_ae[test_index_ae]
      y_train_ae, y_test_ae =
y_full_ae_mapped[train_index_ae],
y_full_ae_mapped[test_index_ae]

      print(f"Train set shape (Autoencoder FS):
{X_train_ae.shape}, Test set shape (Autoencoder FS):
{X_test_ae.shape}")
      print(f"Train label distribution:
{Counter(y_train_ae)}")
      print(f"Test label distribution:
{Counter(y_test_ae)}")

      # Inisialisasi metrik untuk lipatan ini ke nilai
default/NaN.
      accuracy = np.nan
      loss = np.nan
      cm = np.zeros((num_classes_ae,
num_classes_ae), dtype=int) # Inisialisasi matriks
kebingungan menjadi nol.
      roc_auc_ae = np.nan
      fpr_ae = None
      tpr_ae = None
      train_test_time = np.nan
      history_ae = None

      # --- Normalisasi data pelatihan sebelum SMOTE
menggunakan MinMaxScaler ---
      print("Normalizing training data before SMOTE
using MinMaxScaler (Autoencoder FS)...")
      scaler = MinMaxScaler()
      X_train_ae_normalized =
scaler.fit_transform(X_train_ae)
      # Normalisasi data uji menggunakan scaler yang
telah dilatih pada data pelatihan.
      X_test_ae_normalized =
scaler.transform(X_test_ae)
      print("Normalization completed.")

      # --- Terapkan SMOTE pada data pelatihan yang
dinormalisasi untuk lipatan saat ini. ---
      print("Applying SMOTE on normalized training
data (Autoencoder FS)...")
      smote = SMOTE(random_state=42 + fold_idx) #
Gunakan fold_idx untuk konsistensi acak per
pembagian.
      try:
        X_train_ae_smote, y_train_ae_smote =
smote.fit_resample(X_train_ae_normalized,
y_train_ae) # Terapkan SMOTE pada data yang telah
dinormalisasi
        print(f"Train set shape after SMOTE
(Autoencoder FS): {X_train_ae_smote.shape}")
```

```
        print(f"Train label distribution after SMOTE: {Counter(y_train_ae_smote)}")

        # Mengubah bentuk data untuk RNN (timesteps=1) - dipindahkan ke dalam blok try
        X_train_ae_smote_rnn = X_train_ae_smote.reshape((X_train_ae_smote.shape[0], 1, X_train_ae_smote.shape[1]))
        X_test_ae_rnn = X_test_ae_normalized.reshape((X_test_ae_normalized.shape[0], 1, X_test_ae_normalized.shape[1]))

        # --- Pelatihan Jaringan Neural Rekurens (RNN) dengan Autoencoder FS---
        print(f"Training RNN with Autoencoder FS (Fold {fold})...")
        start_time = time.time()
        tf.keras.backend.clear_session()
        rnn_ae_model = build_rnn_model_binary(input_shape=(X_train_ae_smote_rnn.shape[1], X_train_ae_smote_rnn.shape[2]))

        # Latih model dengan Early Stopping dan tampilkan kemajuan epoch.
        history_ae = rnn_ae_model.fit(
            X_train_ae_smote_rnn, y_train_ae_smote,
            epochs=epochs_cv,
            batch_size=batch_size_cv,
            verbose=1, # Tampilkan epoch progress
            validation_data=(X_test_ae_rnn, y_test_ae),
            callbacks=[early_stopping]
        )

        end_time = time.time()
        train_test_time = end_time - start_time

        # Evaluasi model dan hitung metrik setelah pelatihan berhasil.
        loss, accuracy = rnn_ae_model.evaluate(X_test_ae_rnn, y_test_ae, verbose=0)
        y_pred_prob = rnn_ae_model.predict(X_test_ae_rnn)

        # Hitung ROC dan AUC untuk klasifikasi biner
        if len(np.unique(y_test_ae)) < 2:
            print(f"  Mengabaikan perhitungan ROC AUC untuk lipatan {fold} - Autoencoder FS karena hanya ada satu kelas dalam set uji..")
            fpr_ae, tpr_ae, roc_auc_ae = None, None, np.nan
        else:
            fpr_ae, tpr_ae, _ = roc_curve(y_test_ae, y_pred_prob)
            roc_auc_ae = auc(fpr_ae, tpr_ae)
        y_pred = (y_pred_prob > 0.5).astype("int32")
        cm = confusion_matrix(y_test_ae, y_pred, labels=all_labels_mapped_ae)

    except ValueError as e:
        print(f"Could not apply SMOTE for Fold {fold} (Autoencoder FS): {e}")
        print("Skipping training for this fold.")
        # Metrik tetap dalam keadaan terinisialisasi. (NaN/zeros/None)

    scenario_name = 'RNN_AE_FS'

    results_ae[scenario_name]['accuracy'].append(accuracy)
    results_ae[scenario_name]['loss'].append(loss)

    results_ae[scenario_name]['conf_matrix'].append(cm)

    results_ae[scenario_name]['roc_auc'].append(roc_auc_ae)
    results_ae[scenario_name]['fpr'].append(fpr_ae)

    results_ae[scenario_name]['tpr'].append(tpr_ae)

    results_ae[scenario_name]['time'].append(train_test_time)

    results_ae[scenario_name]['history'].append(history_ae.history if history_ae is not None else None) # Pastikan riwayat disimpan atau Tidak
    print(f"  Completed Fold {fold} in {train_test_time:.2f} seconds. Test Accuracy: {accuracy:.4f}, Test Loss: {loss:.4f}")

    # --- Simpan Hasil Sementara Setelah Setiap Lipatan ---
    print(f"Saving intermediate results after Fold {fold}...")
    try:
        # Konversi array numpy menjadi daftar untuk serialisasi JSON, termasuk penanganan nilai None dan NaN.
        results_to_save = {}
        scenario_name_save = 'RNN_AE_FS'
        data_to_save = results_ae[scenario_name_save]
        results_to_save[scenario_name_save] = {
            'accuracy': data_to_save['accuracy'],
            'loss': data_to_save['loss'],
            'conf_matrix': [cm_item.tolist() if cm_item is not None else None for cm_item in data_to_save['conf_matrix']], # Convert numpy array to list, handle None
            'roc_auc': data_to_save['roc_auc'],
```

```
            'fpr': [f.tolist() if f is not None else None for f
in data_to_save['fpr']], # Convert numpy array to list,
handle None
            'tpr': [t.tolist() if t is not None else None for
t in data_to_save['tpr']], # Convert numpy array to
list, handle None
            'time': data_to_save['time'],
            'history': data_to_save['history'] # Sejarah
sudah ditentukan sebagai daftar atau None.
        }

        with open(intermediate_results_path_ae, 'w')
as f:
            json.dump(results_to_save, f)
        print("Intermediate results saved.")
    except Exception as e_save:
        print(f"Error saving intermediate results:
{e_save}")

  print("\n--- Cross-Validation for Autoencoder FS
Completed ---")

  # --- Menggabungkan dan Melaporkan Hasil untuk
Autoencoder FS ---
  print("\n--- Aggregated Results for Autoencoder FS
(Mean +/- Std Dev) ---")

  scenario_name = 'RNN_AE_FS'
  result_data = results_ae[scenario_name]

  # Hitung rata-rata dan simpangan baku untuk
setiap metrik di seluruh lipatan, dengan menangani
nilai NaN.
  avg_accuracy =
np.nanmean(result_data['accuracy']) if
result_data['accuracy'] else np.nan
  std_accuracy = np.nanstd(result_data['accuracy'])
if result_data['accuracy'] else np.nan
  avg_loss = np.nanmean(result_data['loss']) if
result_data['loss'] else np.nan
  std_loss = np.nanstd(result_data['loss']) if
result_data['loss'] else np.nan

  valid_roc_auc = [auc for auc in
result_data['roc_auc'] if auc is not None and not
np.isnan(auc)]
  avg_roc_auc = np.mean(valid_roc_auc) if
valid_roc_auc else np.nan
  std_roc_auc = np.std(valid_roc_auc) if
valid_roc_auc else np.nan

  valid_total_times = [t for t in result_data['time'] if t
is not None and not np.isnan(t)]
  avg_total_time = np.mean(valid_total_times) if
valid_total_times else np.nan
  std_total_time = np.std(valid_total_times) if
valid_total_times else np.nan

  print(f"\nScenario: {scenario_name}")
  print(f"  Average Test Accuracy: {avg_accuracy:.4f}
+/- {std_accuracy:.4f}")
  print(f"  Average Test Loss: {avg_loss:.4f} +/-
{std_loss:.4f}")
  print(f"  Average ROC AUC: {avg_roc_auc:.4f} +/-
{std_roc_auc:.4f}" if not np.isnan(avg_roc_auc) else "
Average ROC AUC: N/A")
  print(f"  Average Total Training/Testing Time:
{avg_total_time:.4f} +/- {std_total_time:.4f} seconds"
if not np.isnan(avg_total_time) else "  Average Total
Training/Testing Time: N/A")

  # --- Save Results for Autoencoder FS ---
  results_save_path_ae =
'/content/drive/MyDrive/DDOS2019/normalisai_trai
n/cv_results_ae_only.xlsx' # Specific path for
Autoencoder only results
```

**#GABUNGAN LVQ DAN AUTOENCODER**

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import time
import warnings
from sklearn.metrics import confusion_matrix,
roc_curve, auc, ConfusionMatrixDisplay
from tensorflow.keras.callbacks import EarlyStopping
import os # Import os
from google.colab import drive # Import drive
import tensorflow as tf # Import tensorflow
from tensorflow.keras.models import Sequential #
Import Sequential
from tensorflow.keras.layers import LSTM, Dropout,
Dense # Import layers
from tensorflow.keras.optimizers import Adam #
Import Adam
from tensorflow.keras import Input # Import Input
from sklearn.model_selection import StratifiedKFold
# Import StratifiedKFold
from imblearn.over_sampling import SMOTE #
Import SMOTE
from collections import Counter # Import Counter
import random # Import random
import seaborn as sns # Import seaborn
import json # Import json
from sklearn.preprocessing import MinMaxScaler #
Import MinMaxScaler for Min-Max normalization

warnings.filterwarnings('ignore')

# Pastikan reproduktifitas
seed_value = 42
np.random.seed(seed_value)
random.seed(seed_value)
```

```
tf.random.set_seed(seed_value)

# Pasang Google Drive jika belum terpasang.
try:
  drive.mount('/content/drive',
force_remount=True)
  print("Google Drive mounted.")
except Exception as e:
  print(f"Error mounting Google Drive: {e}")

# Tentukan jalur file untuk dataset Hybrid
file_path_hybrid =
'/content/drive/MyDrive/DDOS2019/normalisai_trai
n/Dataset2019_FS_Hybrid.csv'

df_hybrid = None
X_full_hybrid = None
y_full_hybrid = None

all_labels_hybrid = None
all_labels_mapped_hybrid = None
num_classes_hybrid = 0

# Muat dataset Hybrid dan periksa keberadaan file.
if os.path.exists(file_path_hybrid):
  try:
    df_hybrid = pd.read_csv(file_path_hybrid)
    X_full_hybrid = df_hybrid.drop('Label',
axis=1).values
    y_full_hybrid = df_hybrid['Label'].values
    print(f"Successfully loaded: {file_path_hybrid}")
  except Exception as e:
    print(f"Error loading {file_path_hybrid}: {e}")
else:
  print(f"File not found: {file_path_hybrid}")

# Lanjutkan hanya jika dataset Hybrid telah dimuat
dengan sukses.
if X_full_hybrid is not None and y_full_hybrid is not
None:
  all_labels_hybrid = np.unique(y_full_hybrid)
  num_classes_hybrid = len(all_labels_hybrid)

  # Mapping label ke bilangan bulat 0 dan 1 untuk
klasifikasi biner
  label_mapping_hybrid = {label: i for i, label in
enumerate(sorted(all_labels_hybrid))}
  y_full_hybrid_mapped =
np.array([label_mapping_hybrid[label] for label in
y_full_hybrid])

  all_labels_mapped_hybrid =
np.unique(y_full_hybrid_mapped)

  # Tentukan fungsi pembangun model RNN untuk
klasifikasi biner.
  def build_rnn_model_binary(input_shape):
    model = Sequential()
    model.add(Input(shape=input_shape))
    model.add(LSTM(units=64,
return_sequences=True))
    model.add(Dropout(0.2))
    model.add(LSTM(units=32,
return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(units=16, activation='relu'))
    model.add(Dense(units=1, activation='sigmoid'))
    optimizer = Adam()
    model.compile(optimizer=optimizer,
loss='binary_crossentropy', metrics=['accuracy'])
    return model

  # --- Cross-Validation Setup ---
  n_splits = 10
  skf_hybrid = StratifiedKFold(n_splits=n_splits,
shuffle=True, random_state=42)

  print(f"\nStarting {n_splits}-fold cross-validation
for Hybrid FS...")

  # --- Define Save Path for Intermediate Results ---
  # Simpan ke Google Drive untuk penyimpanan
permanen - jalur khusus untuk skenario hibrid
  intermediate_results_path_hybrid =
'/content/drive/MyDrive/DDOS2019/normalisai_trai
n/intermediate_cv_results_hybrid_only.json'

  # --- Muat Hasil yang Sudah Ada jika Tersedia ---
  results_hybrid = {
    'RNN_Hybrid_FS': {'accuracy': [], 'loss': [],
'conf_matrix': [], 'roc_auc': [], 'fpr': [], 'tpr': [], 'time':
[], 'history': []}
  }
  start_fold_hybrid = 0

  if
os.path.exists(intermediate_results_path_hybrid):
    print(f"Loading existing results from:
{intermediate_results_path_hybrid}")
    try:
      with open(intermediate_results_path_hybrid,
'r') as f:
        loaded_results = json.load(f)

      scenario_name = 'RNN_Hybrid_FS'
      if scenario_name in loaded_results:
        data = loaded_results[scenario_name]
        results_hybrid[scenario_name]['accuracy'] =
data.get('accuracy', [])
        results_hybrid[scenario_name]['loss'] =
data.get('loss', [])
        # Konversikan daftar kembali menjadi array
numpy untuk matriks kebingungan.
```

```
results_hybrid[scenario_name]['conf_matrix'] =
[np.array(cm) for cm in data.get('conf_matrix', [])]
            results_hybrid[scenario_name]['roc_auc'] =
data.get('roc_auc', [])
            # Konversikan daftar kembali menjadi array
numpy untuk fpr dan tpr
            results_hybrid[scenario_name]['fpr'] =
[np.array(f) if f is not None else None for f in
data.get('fpr', [])]
            results_hybrid[scenario_name]['tpr'] =
[np.array(t) if t is not None else None for t in
data.get('tpr', [])]
            results_hybrid[scenario_name]['time'] =
data.get('time', [])
            results_hybrid[scenario_name]['history'] =
data.get('history', [])

        # Tentukan lipatan awal
        completed_folds =
len(results_hybrid[scenario_name]['accuracy'])
        start_fold_hybrid = completed_folds
        print(f"Resuming from Fold {start_fold_hybrid
+ 1}/{n_splits}")

      except Exception as e:
        print(f"Error loading results from
{intermediate_results_path_hybrid}: {e}")
        print("Starting from the beginning.")
        # Reset hasil jika proses pemuatan gagal
        results_hybrid = {
          'RNN_Hybrid_FS': {'accuracy': [], 'loss': [],
'conf_matrix': [], 'roc_auc': [], 'fpr': [], 'tpr': [], 'time':
[], 'history': []}
        }
        start_fold_hybrid = 0
  else:
    print("No existing results found. Starting from
the beginning.")

  # Parameters for RNN-LSTM Training
  epochs_cv = 50
  batch_size_cv = 64
  early_stopping_patience = 10

  # Tentukan Early Stopping callback
  early_stopping = EarlyStopping(monitor='val_loss',
patience=early_stopping_patience,
restore_best_weights=True, verbose=1)

  # --- Cross-Validation Loop for Hybrid FS ---
  # Lalui setiap pembagian dan lewati lipatan yang
sudah selesai.
  splits_hybrid = list(skf_hybrid.split(X_full_hybrid,
y_full_hybrid_mapped))

  for fold_idx in range(start_fold_hybrid, n_splits):
    fold = fold_idx + 1 # Sesuaikan jumlah lipatan
menjadi berbasis 1.
    train_index_hybrid, test_index_hybrid =
splits_hybrid[fold_idx] # Dapatkan indeks untuk
lipatan saat ini

    print(f"\n--- Fold {fold}/{n_splits} ---")

    X_train_hybrid, X_test_hybrid =
X_full_hybrid[train_index_hybrid],
X_full_hybrid[test_index_hybrid]
    y_train_hybrid, y_test_hybrid =
y_full_hybrid_mapped[train_index_hybrid],
y_full_hybrid_mapped[test_index_hybrid]

    print(f"Train set shape (Hybrid FS):
{X_train_hybrid.shape}, Test set shape (Hybrid FS):
{X_test_hybrid.shape}")
    print(f"Train label distribution:
{Counter(y_train_hybrid)}")
    print(f"Test label distribution:
{Counter(y_test_hybrid)}")

    # Inisialisasi metrik untuk lipatan ini ke nilai
default/NaN.
    accuracy = np.nan
    loss = np.nan
    cm = np.zeros((num_classes_hybrid,
num_classes_hybrid), dtype=int) # Inisialisasi matriks
kebingungan menjadi nol.
    roc_auc_hybrid = np.nan
    fpr_hybrid = None
    tpr_hybrid = None
    train_test_time = np.nan
    history_hybrid = None

    # --- Normalize training data sebelum SMOTE
using MinMaxScaler ---
    print("Normalizing training data sebelum SMOTE
using MinMaxScaler (Hybrid FS)...")
    scaler = MinMaxScaler()
    X_train_hybrid_normalized =
scaler.fit_transform(X_train_hybrid)
    # NNormalisasi data uji menggunakan scaler
yang telah disesuaikan pada data pelatihan.
    X_test_hybrid_normalized =
scaler.transform(X_test_hybrid)
    print("Normalization completed.")

    # --- Terapkan SMOTE pada data pelatihan yang
dinormalisasi untuk lipatan saat ini.---
    print("Applying SMOTE on normalized training
data (Hybrid FS)...")
    smote = SMOTE(random_state=42 + fold_idx) #
Gunakan fold_idx untuk konsistensi acak per
pembagian.
    try:
```

```
        X_train_hybrid_smote, y_train_hybrid_smote = smote.fit_resample(X_train_hybrid_normalized, y_train_hybrid) # Terapkan SMOTE pada data yang telah dinormalisasi
        print(f"Train set shape after SMOTE (Hybrid FS): {X_train_hybrid_smote.shape}")
        print(f"Train label distribution after SMOTE: {Counter(y_train_hybrid_smote)}")

        # Mengubah bentuk data untuk RNN (timesteps=1) - dipindahkan ke dalam blok try
        X_train_hybrid_smote_rnn = X_train_hybrid_smote.reshape((X_train_hybrid_smote.shape[0], 1, X_train_hybrid_smote.shape[1]))
        X_test_hybrid_rnn = X_test_hybrid_normalized.reshape((X_test_hybrid_normalized.shape[0], 1, X_test_hybrid_normalized.shape[1]))

        # --- Training RNN with Hybrid FS ---
        print(f"Training RNN with Hybrid FS (Fold {fold})...")
        start_time = time.time()
        tf.keras.backend.clear_session()
        rnn_hybrid_model = build_rnn_model_binary(input_shape=(X_train_hybrid_smote_rnn.shape[1], X_train_hybrid_smote_rnn.shape[2]))

        # Latih model dengan Early Stopping dan tampilkan kemajuan epoch.
        history_hybrid = rnn_hybrid_model.fit(
            X_train_hybrid_smote_rnn, y_train_hybrid_smote,
            epochs=epochs_cv,
            batch_size=batch_size_cv,
            verbose=1, # tampilkan epoch progress
            validation_data=(X_test_hybrid_rnn, y_test_hybrid),
            callbacks=[early_stopping]
        )

        end_time = time.time()
        train_test_time = end_time - start_time

        # Evaluasi model dan hitung metrik setelah pelatihan berhasil.
        loss, accuracy = rnn_hybrid_model.evaluate(X_test_hybrid_rnn, y_test_hybrid, verbose=0)
        y_pred_prob = rnn_hybrid_model.predict(X_test_hybrid_rnn)

        # Hitung ROC dan AUC untuk klasifikasi biner
        if len(np.unique(y_test_hybrid)) < 2:
            print(f"  Skipping ROC AUC calculation for Fold {fold} - Hybrid FS due to only one class in test set.")
            fpr_hybrid, tpr_hybrid, roc_auc_hybrid = None, None, np.nan
        else:
            fpr_hybrid, tpr_hybrid, _ = roc_curve(y_test_hybrid, y_pred_prob)
            roc_auc_hybrid = auc(fpr_hybrid, tpr_hybrid)

        y_pred = (y_pred_prob > 0.5).astype("int32")
        cm = confusion_matrix(y_test_hybrid, y_pred, labels=all_labels_mapped_hybrid)

    except ValueError as e:
        print(f"Could not apply SMOTE for Fold {fold} (Hybrid FS): {e}")
        print("Skipping training for this fold.")
        # Metrik tetap dalam keadaan awal (NaN/nol/None)


    scenario_name = 'RNN_Hybrid_FS'

results_hybrid[scenario_name]['accuracy'].append(accuracy)

results_hybrid[scenario_name]['loss'].append(loss)

results_hybrid[scenario_name]['conf_matrix'].append(cm)

results_hybrid[scenario_name]['roc_auc'].append(roc_auc_hybrid)

results_hybrid[scenario_name]['fpr'].append(fpr_hybrid)

results_hybrid[scenario_name]['tpr'].append(tpr_hybrid)

results_hybrid[scenario_name]['time'].append(train_test_time)

results_hybrid[scenario_name]['history'].append(history_hybrid.history if history_hybrid is not None else None) # Ensure history is saved or None
    print(f"  Completed Fold {fold} in {train_test_time:.2f} seconds. Test Accuracy: {accuracy:.4f}, Test Loss: {loss:.4f}")

    # --- Simpan Hasil Sementara Setelah Setiap Lipatan ---
    print(f"Saving intermediate results after Fold {fold}...")
    try:
```

```
        # Konversi array numpy menjadi daftar untuk serialisasi JSON, termasuk penanganan nilai None dan NaN.
        results_to_save = {}
        scenario_name_save = 'RNN_Hybrid_FS'
        data_to_save = results_hybrid[scenario_name_save]
        results_to_save[scenario_name_save] = {
            'accuracy': data_to_save['accuracy'],
            'loss': data_to_save['loss'],
            'conf_matrix': [cm_item.tolist() if cm_item is not None else None for cm_item in data_to_save['conf_matrix']], # Convert numpy array to list, handle None
            'roc_auc': data_to_save['roc_auc'],
            'fpr': [f.tolist() if f is not None else None for f in data_to_save['fpr']], # Convert numpy array to list, handle None
            'tpr': [t.tolist() if t is not None else None for t in data_to_save['tpr']], # Convert numpy array to list, handle None
            'time': data_to_save['time'],
            'history': data_to_save['history'] # Sejarah sudah ditentukan sebagai daftar atau None.
        }

        with open(intermediate_results_path_hybrid, 'w') as f:
            json.dump(results_to_save, f)
        print("Intermediate results saved.")
    except Exception as e_save:
        print(f"Error saving intermediate results: {e_save}")


  print("\n--- Cross-Validation for Hybrid FS Completed ---")

  # --- Mengumpulkan dan Melaporkan Hasil untuk Sistem Keuangan Hibrida ---
  print("\n--- Aggregated Results for Hybrid FS (Mean +/- Std Dev) ---")

  scenario_name = 'RNN_Hybrid_FS'
  result_data = results_hybrid[scenario_name]

  # Hitung rata-rata dan simpangan baku untuk setiap metrik di seluruh lipatan, dengan menangani nilai NaN.
  avg_accuracy = np.nanmean(result_data['accuracy']) if result_data['accuracy'] else np.nan
  std_accuracy = np.nanstd(result_data['accuracy']) if result_data['accuracy'] else np.nan
  avg_loss = np.nanmean(result_data['loss']) if result_data['loss'] else np.nan
  std_loss = np.nanstd(result_data['loss']) if result_data['loss'] else np.nan

  valid_roc_auc = [auc for auc in result_data['roc_auc'] if auc is not None and not np.isnan(auc)]
  avg_roc_auc = np.mean(valid_roc_auc) if valid_roc_auc else np.nan
  std_roc_auc = np.std(valid_roc_auc) if valid_roc_auc else np.nan

  valid_total_times = [t for t in result_data['time'] if t is not None and not np.isnan(t)]
  avg_total_time = np.mean(valid_total_times) if valid_total_times else np.nan
  std_total_time = np.std(valid_total_times) if valid_total_times else np.nan

  print(f"\nScenario: {scenario_name}")
  print(f"  Average Test Accuracy: {avg_accuracy:.4f} +/- {std_accuracy:.4f}")
  print(f"  Average Test Loss: {avg_loss:.4f} +/- {std_loss:.4f}")
  print(f"  Average ROC AUC: {avg_roc_auc:.4f} +/- {std_roc_auc:.4f}" if not np.isnan(avg_roc_auc) else "  Average ROC AUC: N/A")
   print(f"  Average Total Training/Testing Time: {avg_total_time:.4f} +/- {std_total_time:.4f} seconds" if not np.isnan(avg_total_time) else "  Average Total Training/Testing Time: N/A")

#TANPA SELEKSI FITUR
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import time
import warnings
from sklearn.metrics import confusion_matrix, roc_curve, auc, ConfusionMatrixDisplay
from tensorflow.keras.callbacks import EarlyStopping
import os # Import os
from google.colab import drive # Import drive
import tensorflow as tf # Import tensorflow
from tensorflow.keras.models import Sequential # Import Sequential
from tensorflow.keras.layers import LSTM, Dropout, Dense # Import layers
from tensorflow.keras.optimizers import Adam # Import Adam
from tensorflow.keras import Input # Import Input
from sklearn.model_selection import StratifiedKFold # Import StratifiedKFold
from imblearn.over_sampling import SMOTE # Import SMOTE
from collections import Counter # Import Counter
import random # Import random
import seaborn as sns # Import seaborn
```

```
import json # Import json
from sklearn.preprocessing import MinMaxScaler #
Import MinMaxScaler for Min-Max normalization

warnings.filterwarnings('ignore')

# Pastikan reproduktifitas
seed_value = 42
np.random.seed(seed_value)
random.seed(seed_value)
tf.random.set_seed(seed_value)

# Pasang Google Drive jika belum terpasang.
try:
  drive.mount('/content/drive',
force_remount=True)
  print("Google Drive mounted.")
except Exception as e:
  print(f"Error mounting Google Drive: {e}")

# Tentukan jalur file untuk dataset Baseline
file_path_baseline =
'/content/drive/MyDrive/DDOS2019/normalisai_trai
n/DDOS2019-Preprosesing.csv'

df_baseline = None
X_full_baseline = None
y_full_baseline = None

all_labels_baseline = None
all_labels_mapped_baseline = None
num_classes_baseline = 0

# Muat dataset Baseline dan periksa keberadaan file.
if os.path.exists(file_path_baseline):
  try:
    df_baseline = pd.read_csv(file_path_baseline)
    X_full_baseline = df_baseline.drop('Label',
axis=1).values
    y_full_baseline = df_baseline['Label'].values
    print(f"Successfully loaded:
{file_path_baseline}")
  except Exception as e:
    print(f"Error loading {file_path_baseline}: {e}")
else:
  print(f"File not found: {file_path_baseline}")

# Lanjutkan hanya jika dataset dasar telah dimuat
dengan sukses.
if X_full_baseline is not None and y_full_baseline is
not None:
  all_labels_baseline = np.unique(y_full_baseline)
  num_classes_baseline = len(all_labels_baseline)

  # Mapping label ke bilangan bulat 0 dan 1 untuk
klasifikasi biner
  label_mapping_baseline = {label: i for i, label in
enumerate(sorted(all_labels_baseline))}
  y_full_baseline_mapped =
np.array([label_mapping_baseline[label] for label in
y_full_baseline])
  all_labels_mapped_baseline =
np.unique(y_full_baseline_mapped)
  # Tentukan fungsi pembangun model RNN untuk
klasifikasi biner.
  def build_rnn_model_binary(input_shape):
    model = Sequential()
    model.add(Input(shape=input_shape))
    model.add(LSTM(units=64,
return_sequences=True))
    model.add(Dropout(0.2))
    model.add(LSTM(units=32,
return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(units=16, activation='relu'))
    model.add(Dense(units=1, activation='sigmoid'))
    optimizer = Adam()
    model.compile(optimizer=optimizer,
loss='binary_crossentropy', metrics=['accuracy'])
    return model

  # --- Cross-Validation Setup ---
  n_splits = 10
  skf_baseline = StratifiedKFold(n_splits=n_splits,
shuffle=True, random_state=42)

  print(f"\nStarting {n_splits}-fold cross-validation
for Baseline...")

  # --- Tentukan jalur penyimpanan untuk hasil
sementara. ---
  # Simpan ke Google Drive untuk penyimpanan
permanen - jalur khusus untuk skenario dasar
  intermediate_results_path_baseline =
'/content/drive/MyDrive/DDOS2019/normalisai_trai
n/intermediate_cv_results_baseline_only.json'

  # --- Muat Hasil yang Sudah Ada jika Tersedia ---
  results_baseline = {
    'RNN_Baseline': {'accuracy': [], 'loss': [],
'conf_matrix': [], 'roc_auc': [], 'fpr': [], 'tpr': [], 'time':
[], 'history': []}
  }
  start_fold_baseline = 0

  if
os.path.exists(intermediate_results_path_baseline):
    print(f"Loading existing results from:
{intermediate_results_path_baseline}")
    try:
      with
open(intermediate_results_path_baseline, 'r') as f:
        loaded_results = json.load(f)
```

```
        scenario_name = 'RNN_Baseline'
        if scenario_name in loaded_results:
            data = loaded_results[scenario_name]
            results_baseline[scenario_name]['accuracy'] = data.get('accuracy', [])
            results_baseline[scenario_name]['loss'] = data.get('loss', [])
            # Konversikan daftar daftar kembali menjadi array numpy untuk matriks kebingungan.
            results_baseline[scenario_name]['conf_matrix'] = [np.array(cm) for cm in data.get('conf_matrix', [])]
            results_baseline[scenario_name]['roc_auc'] = data.get('roc_auc', [])
            # Konversikan daftar daftar kembali menjadi array numpy untuk fpr dan tpr
            results_baseline[scenario_name]['fpr'] = [np.array(f) if f is not None else None for f in data.get('fpr', [])]
            results_baseline[scenario_name]['tpr'] = [np.array(t) if t is not None else None for t in data.get('tpr', [])]
            results_baseline[scenario_name]['time'] = data.get('time', [])
            results_baseline[scenario_name]['history'] = data.get('history', [])

        # Tentukan lipatan awal
        completed_folds = len(results_baseline[scenario_name]['accuracy'])
        start_fold_baseline = completed_folds
        print(f"Resuming from Fold {start_fold_baseline + 1}/{n_splits}")

    except Exception as e:
        print(f"Error loading results from {intermediate_results_path_baseline}: {e}")
        print("Starting from the beginning.")
        # Reset hasil jika proses pemuatan gagal
        results_baseline = {
            'RNN_Baseline': {'accuracy': [], 'loss': [], 'conf_matrix': [], 'roc_auc': [], 'fpr': [], 'tpr': [], 'time': [], 'history': []}
        }
        start_fold_baseline = 0
else:
    print("No existing results found. Starting from the beginning.")

# Parameter untuk Pelatihan RNN-LSTM
epochs_cv = 50
batch_size_cv = 64
early_stopping_patience = 10

# Tentukan Early Stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=early_stopping_patience, restore_best_weights=True, verbose=1)

# --- Lingkaran Validasi Silang untuk Model Dasar ---
# Lalui setiap pembagian dan lewati lipatan yang sudah selesai.
splits_baseline = list(skf_baseline.split(X_full_baseline, y_full_baseline_mapped))

for fold_idx in range(start_fold_baseline, n_splits):
    fold = fold_idx + 1 # Sesuaikan jumlah lipatan menjadi berbasis 1.
    train_index_baseline, test_index_baseline = splits_baseline[fold_idx] # Dapatkan indeks untuk lipatan saat ini

    print(f"\n--- Fold {fold}/{n_splits} ---")

    X_train_baseline, X_test_baseline = X_full_baseline[train_index_baseline], X_full_baseline[test_index_baseline]
    y_train_baseline, y_test_baseline = y_full_baseline_mapped[train_index_baseline], y_full_baseline_mapped[test_index_baseline]

    print(f"Train set shape (Baseline): {X_train_baseline.shape}, Test set shape (Baseline): {X_test_baseline.shape}")
    print(f"Train label distribution: {Counter(y_train_baseline)}")
    print(f"Test label distribution: {Counter(y_test_baseline)}")

    # Inisialisasi metrik untuk lipatan ini ke nilai default/NaN.
    accuracy = np.nan
    loss = np.nan
    cm = np.zeros((num_classes_baseline, num_classes_baseline), dtype=int) # Inisialisasi matriks kebingungan menjadi nol.
    roc_auc_baseline = np.nan
    fpr_baseline = None
    tpr_baseline = None
    train_test_time = np.nan
    history_baseline = None

    # --- Normalisasi data pelatihan sebelum SMOTE menggunakan MinMaxScaler ---
    print("Normalizing training data before SMOTE using MinMaxScaler (Baseline)...")
    scaler = MinMaxScaler()
    X_train_baseline_normalized = scaler.fit_transform(X_train_baseline)
```

```
    # Normalize test data using the scaler fitted on
training data
    X_test_baseline_normalized =
scaler.transform(X_test_baseline)
    print("Normalization completed.")

    # --- Terapkan SMOTE pada data pelatihan yang
dinormalisasi untuk lipatan saat ini. ---
    print("Applying SMOTE on normalized training
data (Baseline)...")
    smote = SMOTE(random_state=42 + fold_idx) #
Gunakan fold_idx untuk konsistensi acak per
pembagian.
    try:
      X_train_baseline_smote,
y_train_baseline_smote =
smote.fit_resample(X_train_baseline_normalized,
y_train_baseline) # Terapkan SMOTE pada data yang
telah dinormalisasi
      print(f"Train set shape after SMOTE (Baseline):
{X_train_baseline_smote.shape}")
      print(f"Train label distribution after SMOTE:
{Counter(y_train_baseline_smote)}")

      # Mengubah bentuk data untuk RNN
(timesteps=1) - dipindahkan ke dalam blok try
      X_train_baseline_smote_rnn =
X_train_baseline_smote.reshape((X_train_baseline_s
mote.shape[0], 1, X_train_baseline_smote.shape[1]))
      X_test_baseline_rnn =
X_test_baseline_normalized.reshape((X_test_baselin
e_normalized.shape[0], 1,
X_test_baseline_normalized.shape[1]))
      # --- Pelatihan RNN dengan Model Dasar ---
      print(f"Training RNN with Baseline (Fold
{fold})...")
      start_time = time.time()
      tf.keras.backend.clear_session()
      rnn_baseline_model =
build_rnn_model_binary(input_shape=(X_train_base
line_smote_rnn.shape[1],
X_train_baseline_smote_rnn.shape[2]))
      # Latih model dengan Early Stopping dan
tampilkan kemajuan epoch.
      history_baseline = rnn_baseline_model.fit(
        X_train_baseline_smote_rnn,
y_train_baseline_smote,
        epochs=epochs_cv,
        batch_size=batch_size_cv,
        verbose=1, # Tampilkan epoch progress
        validation_data=(X_test_baseline_rnn,
y_test_baseline),
        callbacks=[early_stopping]
      )

      end_time = time.time()
      train_test_time = end_time - start_time

      # Evaluasi model dan hitung metrik setelah
pelatihan berhasil.
      loss, accuracy =
rnn_baseline_model.evaluate(X_test_baseline_rnn,
y_test_baseline, verbose=0)
      y_pred_prob =
rnn_baseline_model.predict(X_test_baseline_rnn)

      # Hitung ROC dan AUC untuk klasifikasi biner
      if len(np.unique(y_test_baseline)) < 2:
        print(f"  Skipping ROC AUC calculation for
Fold {fold} - Baseline due to only one class in test
set.")
        fpr_baseline, tpr_baseline,
roc_auc_baseline = None, None, np.nan
      else:
        fpr_baseline, tpr_baseline, _ =
roc_curve(y_test_baseline, y_pred_prob)
        roc_auc_baseline = auc(fpr_baseline,
tpr_baseline)

      y_pred = (y_pred_prob > 0.5).astype("int32")
      cm = confusion_matrix(y_test_baseline,
y_pred, labels=all_labels_mapped_baseline)

    except ValueError as e:
      print(f"Could not apply SMOTE for Fold {fold}
(Baseline): {e}")
      print("Skipping training for this fold.")
      # Metrik tetap dalam keadaan awal
(NaN/nol/None)

    scenario_name = 'RNN_Baseline'

results_baseline[scenario_name]['accuracy'].append(
accuracy)

results_baseline[scenario_name]['loss'].append(loss)

results_baseline[scenario_name]['conf_matrix'].appe
nd(cm)

results_baseline[scenario_name]['roc_auc'].append(
roc_auc_baseline)

results_baseline[scenario_name]['fpr'].append(fpr_b
aseline)

results_baseline[scenario_name]['tpr'].append(tpr_b
aseline)

results_baseline[scenario_name]['time'].append(trai
n_test_time)

results_baseline[scenario_name]['history'].append(hi
```

```
story_baseline.history if history_baseline is not None else None) # Ensure history is saved or None
      print(f"  Completed Fold {fold} in {train_test_time:.2f} seconds. Test Accuracy: {accuracy:.4f}, Test Loss: {loss:.4f}")

      # --- Simpan Hasil Sementara Setelah Setiap Lipatan ---
      print(f"Saving intermediate results after Fold {fold}...")
      try:
        # Konversi array numpy menjadi daftar untuk serialisasi JSON, termasuk penanganan nilai None dan NaN.
        results_to_save = {}
        scenario_name_save = 'RNN_Baseline'
        data_to_save = results_baseline[scenario_name_save]
        results_to_save[scenario_name_save] = {
          'accuracy': data_to_save['accuracy'],
          'loss': data_to_save['loss'],
          'conf_matrix': [cm_item.tolist() if cm_item is not None else None for cm_item in data_to_save['conf_matrix']], # Convert numpy array to list, handle None
          'roc_auc': data_to_save['roc_auc'],
          'fpr': [f.tolist() if f is not None else None for f in data_to_save['fpr']], # Convert numpy array to list, handle None
          'tpr': [t.tolist() if t is not None else None for t in data_to_save['tpr']], # Convert numpy array to list, handle None
          'time': data_to_save['time'],
          'history': data_to_save['history'] # Sejarah sudah ditentukan sebagai daftar atau None.
        }

        with open(intermediate_results_path_baseline, 'w') as f:
          json.dump(results_to_save, f)
        print("Intermediate results saved.")
      except Exception as e_save:
        print(f"Error saving intermediate results: {e_save}")

  print("\n--- Cross-Validation for Baseline Completed ---")

  # --- Mengumpulkan dan Melaporkan Hasil untuk baseline ---
  print("\n--- Aggregated Results for Baseline (Mean +/- Std Dev) ---")

  scenario_name = 'RNN_Baseline'
  result_data = results_baseline[scenario_name]

  # Hitung rata-rata dan simpangan baku untuk setiap metrik di seluruh lipatan, dengan menangani nilai NaN.
  avg_accuracy = np.nanmean(result_data['accuracy']) if result_data['accuracy'] else np.nan
  std_accuracy = np.nanstd(result_data['accuracy']) if result_data['accuracy'] else np.nan
  avg_loss = np.nanmean(result_data['loss']) if result_data['loss'] else np.nan
  std_loss = np.nanstd(result_data['loss']) if result_data['loss'] else np.nan

  valid_roc_auc = [auc for auc in result_data['roc_auc'] if auc is not None and not np.isnan(auc)]
  avg_roc_auc = np.mean(valid_roc_auc) if valid_roc_auc else np.nan
  std_roc_auc = np.std(valid_roc_auc) if valid_roc_auc else np.nan

  valid_total_times = [t for t in result_data['time'] if t is not None and not np.isnan(t)]
  avg_total_time = np.mean(valid_total_times) if valid_total_times else np.nan
  std_total_time = np.std(valid_total_times) if valid_total_times else np.nan

  print(f"\nScenario: {scenario_name}")
  print(f"  Average Test Accuracy: {avg_accuracy:.4f} +/- {std_accuracy:.4f}")
  print(f"  Average Test Loss: {avg_loss:.4f} +/- {std_loss:.4f}")
  print(f"  Average ROC AUC: {avg_roc_auc:.4f} +/- {std_roc_auc:.4f}" if not np.isnan(avg_roc_auc) else "  Average ROC AUC: N/A")
  print(f"  Average Total Training/Testing Time: {avg_total_time:.4f} +/- {std_total_time:.4f} seconds" if not np.isnan(avg_total_time) else "  Average Total Training/Testing Time: N/A")
```