Objectif : Analyse des logs par des techniques d'apprentissage automatique

- Partie 2 : comparaison

Brain [1] : Bidirectional parallel tree

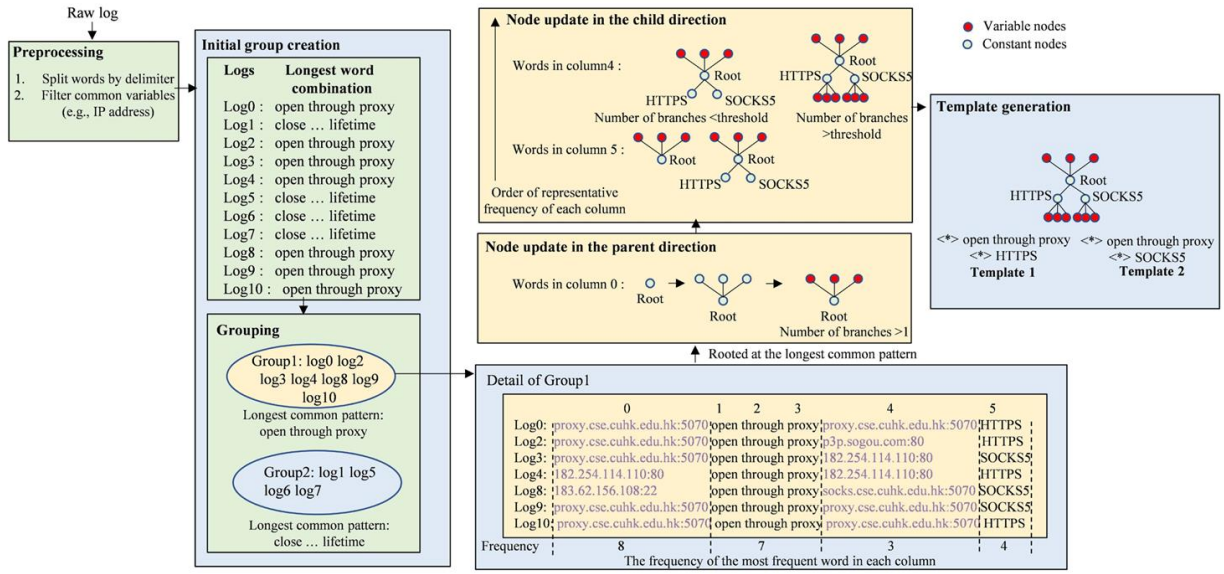
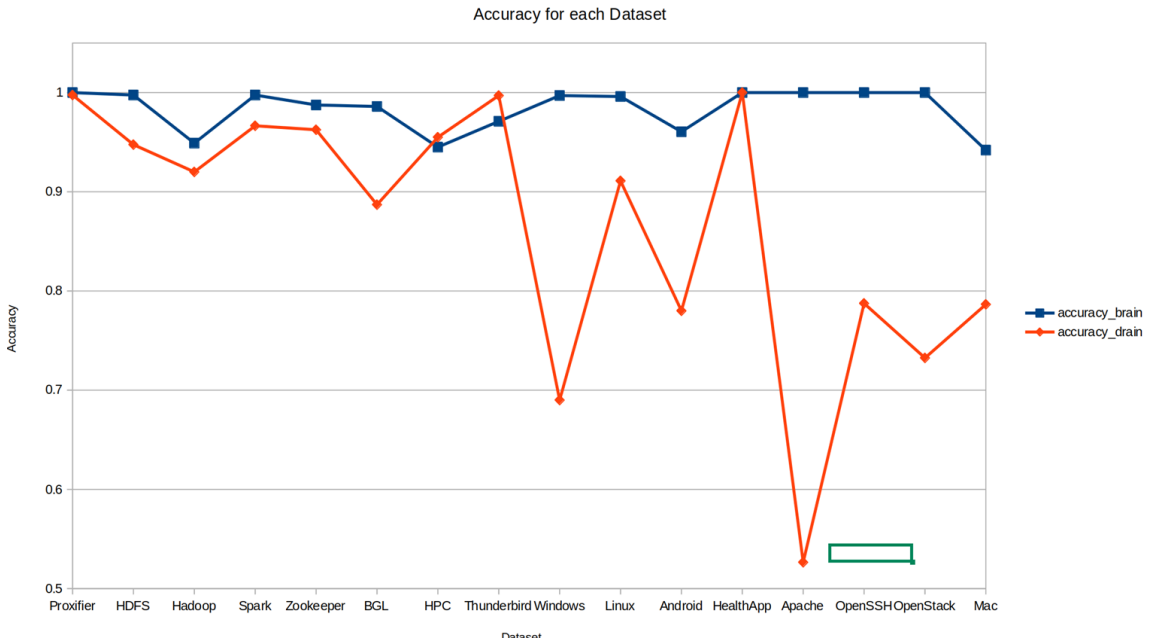


Fig. 3. The workflow of Brain parsing the log messages in Fig. 2.

Beaucoup plus efficaces que Drain [2].



En testant sur tous les logs entiers [3] (sauf Windows, HDFS, BGL, Spark, ThunderBird et Android, trop gros pour mon ordinateur (travail annoncé à 48h sur un serveur par les auteurs de loghub)) soit environ 10min de travail pour ceux-ci :

Overall	evaluation	results
Dataset	F1_measure	Accuracy
Proxifier	0.786614	0.521013
Hadoop	0.834018	0.5627
Zookeeper	0.999873	0.993416
HPC	0.99183	0.800297
Linux	0.999375	0.790184
HealthApp	0.99942	0.979298
Apache	0.999983	0.996537
OpenSSH	0.999988	0.663158
OpenStack	1	1
Mac	0.979953	0.834111

Brain reste très compétitif sur les fichiers complets.
Output : template + log structuré.
Ex :
- template :

EventId	EventTemplate	Occurrences
E42	Receiving block <*> src: <*> dest: <*>	1723232
E34	BLOCK* NameSystem.allocateBlock: <*> <*>	575061
E31	PacketResponder <*> for block <*> terminating	1706679

- structuré :

Content	EventId	EventTemplate
PacketResponder 1 for block blk_3886504906413966O terminating	E0	PacketResponder <*> for block <*> terminating
PacketResponder 0 for block blk_-6952295868487656571 terminating	E0	PacketResponder <*> for block <*> terminating
BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.73.220:50010 is added to blk_7128370237687728475 size 67108864	E3	BLOCK* NameSystem.addStoredBlock: blockMap updated: <*> is added to <*> size <*>
PacketResponder 2 for block blk_8229193803249955061 terminating	E0	PacketResponder <*> for block <*> terminating
PacketResponder 2 for block blk_-6670958622368987959 terminating	E0	PacketResponder <*> for block <*> terminating
BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.43.115:50010 is added to blk_3050920587428079149 size 67108864	E3	BLOCK* NameSystem.addStoredBlock: blockMap updated: <*> is added to <*> size <*>
BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.203.80:50010 is added to blk_7888946331804732825 size 67108864	E3	BLOCK* NameSystem.addStoredBlock: blockMap updated: <*> is added to <*> size <*>
BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.250.11.85:50010 is added to blk_2377150260128098806 size 67108864	E3	BLOCK* NameSystem.addStoredBlock: blockMap updated: <*> is added to <*> size <*>

Fonctionnement :
- Pour Deeplog [4][5], nécessité de convertir le log structuré en une suite de temps, machine et EventId.

timestamp	machine	event
1		0
2		1
3		0
4		0
5		3
6		3
7		6
8		6
9		3
10		6

J'ai fait un petit script pour faire ça automatiquement :

```
if args.format == "HDFS" or args.format == "Hadoop" or args.format == "OpenSSH":
    with open(args.csvOut, 'w', newline='') as outfile:
        fieldnames = ["timestamp", "machine", "event"]
        writer = csv.DictWriter(outfile, fieldnames=fieldnames)
        writer.writeheader()

        # Loop
        for row in csv_reader:
            event_id = row[7].replace("E", "")

            timestamp = row[0]

            writer.writerow({"timestamp": timestamp, "machine": "", "event": event_id})
```

et la commande à exécuter pour le lancer :

"'python test.py ./data/HDFS_100k.log_structured.csv test_HDFS.csv HDFS"'

Pour les fichiers trop gros, j'ai fait un autre script pour ne récupérer que les n premières lignes du fichier de log :

```
def main() -> None:
    """Main method executed when program is run."""
    # Parse arguments
    args = parse_args()

    try:
        # read
        with open(args.logIn, 'r') as i_f:
            lines = i_f.readlines()

        # extract
        n = int(args.n)
        extracted_lines = lines[:n]

        # write
        with open(args.logOut, 'w') as o_f:
            o_f.writelines(extracted_lines)

        print(f"Extracted {args.n} lines from {args.logIn} and saved to {args.logOut}")
    except:
        print(f"Error: file '{args.logIn}' not found.")
```

Conversion du fichier txt fournis dans le GitHub en fichier csv comparable à tous les logs différents.

timestamp	machine	event
0	0	5
1	0	5
2	0	5
3	0	22
4	0	11
5	0	9
6	0	11

Tout est prêt pour lancer DeepLog.

Résultats sur DeepCase (gauche) et DeepLog (droite) :

	precision	recall	f1-score	support
0	0.4753	0.4538	0.4643	92151
1	0.7085	0.7285	0.7184	327893
2	0.7280	0.5703	0.6395	273499
3	0.8193	0.9993	0.9004	1331195
4	0.3796	0.4514	0.4124	2107
5	0.9933	0.9885	0.9909	1329087
6	0.9046	0.9407	0.9223	1329087
7	0.0000	0.0000	0.0000	2042
8	0.6956	0.5808	0.6330	2042
9	0.0000	0.0000	0.0000	172
10	0.9978	0.9994	0.9986	1084158
11	0.9937	0.3408	0.5076	443029
12	0.8796	0.9237	0.9011	1082301
13	0.0000	0.0000	0.0000	2042
14	0.9423	0.9102	0.9260	1332955
15	0.0000	0.0000	0.0000	2
16	0.0000	0.0000	0.0000	10
accuracy			0.9012	8633772
macro avg	0.5599	0.5228	0.5303	8633772
weighted avg	0.9058	0.9012	0.8934	8633772

Classification report - predictions

	precision	recall	f1-score	support
0	0.9998	0.9999	0.9998	92151
1	0.9999	1.0000	0.9999	327893
2	1.0000	1.0000	1.0000	273499
3	1.0000	1.0000	1.0000	1331195
4	0.9976	0.9867	0.9921	2107
5	1.0000	1.0000	1.0000	1329087
6	1.0000	1.0000	1.0000	1329087
7	0.9965	0.9897	0.9931	2042
8	0.9925	0.9775	0.9849	2042
9	1.0000	1.0000	1.0000	172
10	1.0000	1.0000	1.0000	1084158
11	1.0000	1.0000	1.0000	443029
12	1.0000	1.0000	1.0000	1082301
13	0.9956	0.9966	0.9961	2042
14	0.9999	1.0000	1.0000	1332955
15	0.0000	0.0000	0.0000	2
16	1.0000	1.0000	1.0000	10
accuracy			1.0000	8633772
macro avg	0.9401	0.9383	0.9392	8633772
weighted avg	1.0000	1.0000	1.0000	8633772

DeepLog semble meilleur (anormal?) mais DeepCase était 3x plus rapide.
Test sur mon jeu de HDFS en 1 million de ligne :

difficile à interpréter, score F1 très bon sur la moyenne pondérée mais beaucoup moins sur la moyenne globale. Complètement dans les choux sur DeepCase, moyenne inférieure à 1 %...

Test de DeepLog sur tous les jeux de données :

	DeepLog results		
Dataset	F1		avg
Proxifier	1	0.9713	0.98565
Hadoop	0.4241	0.4241	0.4241
Zookeeper	0.6168	0.5861	0.60145
HPC	0.1238	0.1286	0.1262
Linux	0.2111	0.2143	0.2127
HealthApp	0.8155	0.8016	0.80855
Apache	0.861	0.8598	0.8604
OpenSSH	0.6154	0.6154	0.6154
OpenStack	0.6038	0.5916	0.5977
Mac	0.1292	0.1447	0.13695
HDFS_1M	0.8276	0.8278	0.8277
HDFS_100k	0.9967	0.9971	0.9969

Naturellement, sur le set sur lequel DeepLog a été conçu (HDFS), ça fonctionne très bien, sur les autres, les résultats sont plus mitigés.

Attention, gros inconvénient : temps de calcul très long en fonction de la longueur du fichier (O(n), n le nombre de ligne contenues dans le fichier).

Test sur DeepCase :
impossible, résultats proche de 0 pour tous…
A investiguer, ne calcule qu'une seule valeur sur toutes, faussant les résultats, je ne sais pas pourquoi…

Tests sur LogAnomaly [6] et LogRobust [7] impossibles, pas de code source disponible en ligne.
Mais je suis tombé sur une étude [8] qui propose de comparer DeepLog, LogAnomaly, LogBERT, PLELog, CNN, LogRobust et NeuralLog.

Classification report - predictions

	precision	recall	f1-score	support
0	0.5997	1.0000	0.7498	150346
1	0.0000	0.0000	0.0000	2
2	0.0000	0.0000	0.0000	2
3	1.0000	1.0000	1.0000	50529
4	1.0000	1.0000	1.0000	15034
5	1.0000	1.0000	1.0000	149373
6	1.0000	1.0000	1.0000	34957
7	0.0000	0.0000	0.0000	108
8	1.0000	1.0000	1.0000	149368
9	0.0000	0.0000	0.0000	2
10	1.0000	1.0000	1.0000	149536
11	0.0000	0.0000	0.0000	2
12	0.0000	0.0000	0.0000	124
13	1.0000	1.0000	1.0000	9
14	0.0000	0.0000	0.0000	9
15	1.0000	1.0000	1.0000	489
16	0.0000	0.0000	0.0000	108
17	0.0000	0.0000	0.0000	108
18	0.0000	0.0000	0.0000	3
19	0.0000	0.0000	0.0000	3967
20	0.0000	0.0000	0.0000	91650
21	0.0000	0.0000	0.0000	2
23	0.0000	0.0000	0.0000	4067
24	0.0000	0.0000	0.0000	105
25	0.0000	0.0000	0.0000	82
26	0.0000	0.0000	0.0000	2
27	0.0000	0.0000	0.0000	3
28	0.0000	0.0000	0.0000	3
29	0.0000	0.0000	0.0000	4
30	0.0000	0.0000	0.0000	2
31	0.0000	0.0000	0.0000	2
32	0.0000	0.0000	0.0000	2
accuracy			0.8746	800000
macro avg	0.2687	0.2812	0.2734	800000
weighted avg	0.7993	0.8746	0.8275	800000

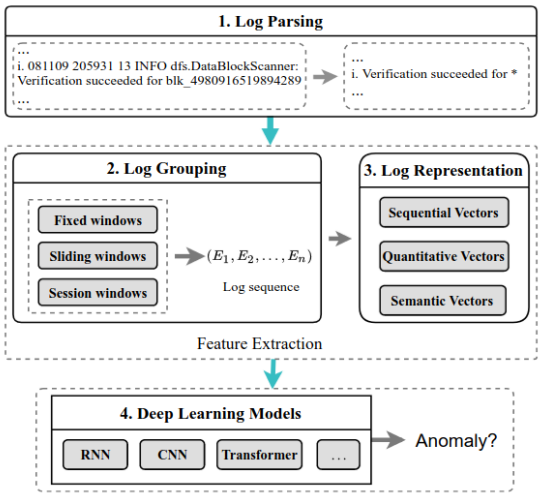


Figure 2: Log-based Anomaly Detection Workflow with Deep Learning: The Common Workflow

Résultats annoncés sur du HDFS :

Table 4: Results of models with session grouping on BGL and HDFS datasets

Model	BGL				HDFS			
	Prec	*Rec*	*Spec*	*F1*	*Prec*	*Rec*	*Spec*	*F1*
DeepLog	0.997	1.0	0.997	0.998	0.835	0.994	0.994	0.908
LogAnomaly	0.997	1.0	0.998	0.999	0.886	0.893	0.961	0.966
PLELog	0.995	0.992	0.996	0.994	0.893	0.979	0.996	0.934
LogRobust	1.0	1.0	1.0	1.0	0.961	1.0	0.989	0.980
CNN	1.0	1.0	1.0	1.0	0.966	1.0	0.991	0.982

Mes résultats sur du HDFS :

		DeepLog		
	Prec	Rec	Acc	F1
1	0.9036	0.8197	0.9938	0.8596
2	0.903	0.8011	0.9933	0.849
3	0.903	0.8278	0.994	0.8638
4	0.9273	0.8226	0.9943	0.8718
5	0.9264	0.9437	0.9973	0.935
avg	0.91266	0.84298	0.99454	0.87584

Impossible d'exécuter les autres, à investiguer.

Je vais plutôt essayer de tester chaque algorithme individuellement.
→ LogBERT [9] :

VHM :

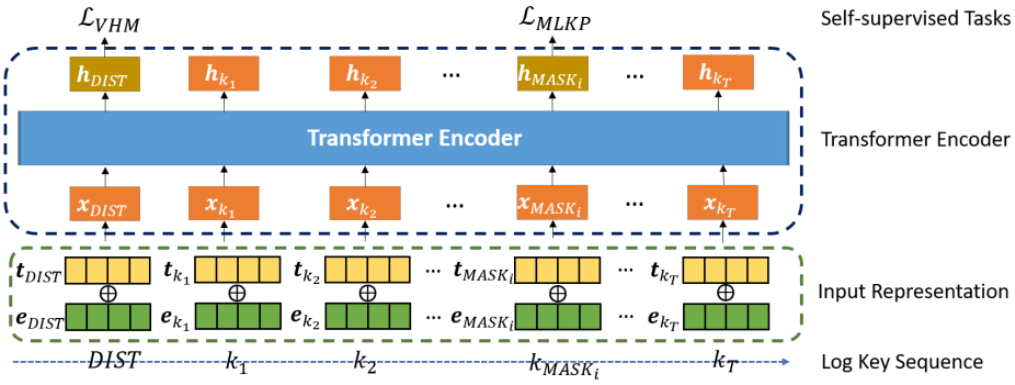


Fig. 1: The overview of LogBERT

Volume of Hypersphere Minimization
MLKP : Masked Log Key Protection

Avantage : pas de template nécessaire comme les autres
Inconvéniant : label des anomalies nécessaire
→ /!\ que HDFS disponible

première étape :
- vectorisation grâce à BERT

fichiers identiques à ce que DeepLog propose dans son étude donc pourquoi ne pas le tester dans DeepLog et DeepCase ?

Classification report - predictions

	precision	recall	f1-score	support
0	0.9113	0.7862	0.8442	175081
1	0.8788	0.9200	0.8989	174569
2	0.6154	0.8773	0.7234	174520
3	0.9835	0.9827	0.9831	174520
4	0.8385	0.8176	0.8279	82539
5	0.0000	0.0000	0.0000	58360
6	0.6412	0.8461	0.7295	31466
7	0.9546	0.5454	0.6942	15614
8	0.9970	0.9870	0.9920	3397
9	0.9987	0.6662	0.7992	3388
10	0.0000	0.0000	0.0000	23
11	0.0000	0.0000	0.0000	17
12	0.0000	0.0000	0.0000	11
13	0.0000	0.0000	0.0000	11
14	0.0000	0.0000	0.0000	6
15	0.0000	0.0000	0.0000	6
accuracy			0.8182	893528
macro avg	0.4887	0.4643	0.4683	893528
weighted avg	0.7868	0.8182	0.7954	893528

Classification report - anomalies

	precision	recall	f1-score	support
Normal	0.9672	0.9999	0.9833	893528
Anomaly	0.9548	0.0578	0.1090	32140
accuracy			0.9672	925668
macro avg	0.9610	0.5288	0.5461	925668
weighted avg	0.9668	0.9672	0.9529	925668

Sur DeepLog, moyenne du F1 à 0.58253 après 10 essais.

Même jeu de donnée sur DeepCase :
moyenne du F1 à 0.8360

Comment faire pour la suite ?
→ attention, ne fonctionne que sur du HDFS car il faut connaître à l'avance toutes les anomalies (référencées).

Retour sur LogBERT :

résultats annoncés :

Method	HDFS		
	Precision	Recall	F-1 score
PCA	5.89	100.00	11.12
iForest	53.60	69.41	60.49
OCSVM	2.54	100.00	4.95
LogCluster	99.26	37.08	53.99
DeepLog	88.44	69.49	77.34
LogAnomaly	94.15	40.47	56.19
LogBERT	87.02	78.10	**82.32**

mes résultats sur HDFS 1M :

Method	HDFS		
	Precision	Recall	F-1 score
PCA	3.89	78.73	7.41
iForest	16.61	56.83	25.71
OCSVM	1.11	36.91	2.15
LogCluster	25	0.06	0.11
DeepLog	3.09	100	6.01
LogAnomaly	2.81	72.82	5.42
LogBERT	79.95	18.22	29.67

Résultats très différents du papier, peut-être dû à la taille du fichier 10x inférieure mais le temps serait beaucoup trop long sur le fichier complet.

Problème : non adaptable sur d'autres formats de logs.

Références :

[1] : S. Yu, P. He, N. Chen and Y. Wu, "Brain: Log Parsing With Bidirectional Parallel Tree," in *IEEE Transactions on Services Computing*, vol. 16, no. 5, pp. 3224-3237, Sept.-Oct. 2023, doi: 10.1109/TSC.2023.3270566.

[2] : P. He, J. Zhu, Z. Zheng and M. R. Lyu, "Drain: An Online Log Parsing Approach with Fixed Depth Tree," *2017 IEEE International Conference on Web Services (ICWS)*, Honolulu, HI, USA, 2017, pp. 33-40, doi: 10.1109/ICWS.2017.13.

[3] : Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, Michael R. Lyu. Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics. IEEE International Symposium on Software Reliability Engineering (ISSRE), 2023.

[4] : van Ede, T., Aghakhani, H., Spahn, N., Bortolameotti, R., Cova, M., Continella, A., van Steen, M., Peter, A., Kruegel, C. & Vigna, G. (2022, May). DeepCASE: Semi-Supervised Contextual Analysis of Security Events. In 2022 Proceedings of the IEEE Symposium on Security and Privacy (S&P). IEEE.

[5] : Du, M., Li, F., Zheng, G., & Srikumar, V. (2017). Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS) (pp. 1285-1298).

[6] : W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, et al., "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs.," in IJCAI, vol. 19, pp. 4739–4745, 2019.

[7] : X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, et al., "Robust log-based anomaly detection on unstable log data," in Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 807–817, 2019.

[8] : Le, Van-Hoang and Zhang, Hongyu, "Log-based Anomaly Detection with Deep Learning: How Far Are We?", 2022 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), 2022.

[9] : H. Guo, S. Yuan, and X. Wu, "Logbert: Log anomaly detection via bert," arXiv preprint arXiv:2103.04475, 2021.