

Mémoire de PFE

Formation FIL

IMT Atlantique



Amélioration des performances de détection automatique d'anomalies dans des logs applicatifs

Oscar Gloaguen

Direction Générale des Finances Publiques



Table des matières

Table des matières	i
Remerciements	iii
Résumé	iv
Introduction	1
1 La DGFiP, administration en évolution constante	2
2 Notions techniques	4
2.1 Que sont les logs ?	4
2.2 Vectorisation sémantique	5
2.3 Forêt d'isolation	6
3 L'existant : le projet analyse-logs	8
3.1 Contexte du projet	8
3.2 Le modèle original : DeepLog	9
3.3 Evolutions du projet par Léa	11
4 Le processus scientifique au cœur du projet	13
4.1 État de l'art de l'analyse de logs	13
4.2 Analyse des jeux de données disponibles	18
5 Organisation du projet	22
5.1 La planification et les aléas	22
5.2 Agilité et recherche : une union difficile	22
6 Développement du nouveau projet	23
7 Projection de l'impact du projet	24
7.1 Gain de temps et de productivité	24

7.2 L'humain contre l'algorithme	24
Conclusion	25
Bibliographie	
Glossaire administratif	
Glossaire technique	
Table des figures	
Annexes	

Remerciements

Je souhaite remercier la Direction Générale des Finances Publiques et tout particulièrement Olivier Blanc, pour m'avoir permis d'effectuer mon alternance dans cette équipe et pour m'avoir accompagné tout au long jusqu'au PFE.

Je tiens à remercier personnellement Robin Gries, qui a su être un vrai coéquipier tout au long de ce projet malgré sa complexité.

Je voudrais aussi remercier IMT Atlantique ainsi que ses professeurs, et surtout mon tuteur pédagogique Thomas Ledoux qui s'est mis à ma disposition pour le PFE.

Résumé

Les **logs** sont une source importante de données détaillant le fonctionnement interne d'une application, mais ne sont pourtant que rarement utilisés à leur plein potentiel. Dans ce mémoire, je vais détailler le processus d'évolution d'un outil d'**apprentissage automatique** qui utilise les **logs** pour détecter et même tenter de prévoir des anomalies logicielles. Le projet s'apparentant plus à un projet de recherche, la démarche scientifique sera détaillée, ainsi que les caractéristiques techniques et le déroulement de l'implémentation. L'organisation du projet avec un stagiaire et moi-même sera développée, ainsi que ses impacts humains et économiques.

Abstract

Logs are an important source of data when it comes to the internal workings of software, but they are rarely used to their full potential. In this memoir, I will explain the evolution of a machine learning tool which uses **logs** to detect and even attempt to predict software anomalies. The project being similar to a research project, the scientific protocol will be detailed, as well as the technical characteristics and the course of the implementation. Project management with an intern and myself will be developed, as well as the human and economic impacts of the project.

Mots-clés traitement automatique du langage (TAL), apprentissage automatique, apprentissage profond, analyse de logs applicatifs

Introduction

1-2 pages

Le projet analyse-logs a été développé par deux précédents apprentis de la [Direction Générale des Finances Publiques \(DGFIP\)](#), Rémi puis Léa. Cependant, les performances des algorithmes implémentés n'étant pas satisfaisantes, c'est ici que naît le sujet de ce PFE. Ce mémoire détaille le processus d'évolution de ce projet dans l'objectif d'amélioration des performances.

J'ai travaillé sur cette problématique en binôme avec Robin, un stagiaire en dernière année de master. Ce document touchera aussi sur l'organisation du sujet entre nous ainsi que les bénéfices et difficultés à travailler en équipe.

Ce sujet de PFE s'intègre dans la stratégie d'innovation du SI de la [DGFIP](#). L'objectif est de montrer l'efficacité de ces outils, pour mettre en valeur l'innovation et pousser leur utilisation au sein des bureaux. Dans ce cadre, une structure proche de celle d'un projet de recherche a été suivie. Pour cela, nous avons d'abord composé et étudié un état de l'art des algorithmes de détection d'anomalies existants. Ils utilisent pour la plupart des méthodes d'[apprentissage automatique](#), avec des algorithmes classiques ou de l'[apprentissage profond](#).

Chapitre 1

La DGFIP, administration en évolution constante

La DGFIP est une administration française née de la fusion de la Direction Générale des Impôts (DGI) et de la Direction Générale des Comptes Publics (DGCP) en 2008. Elle hérite alors des missions des deux entités, en faisant un service public très étendu, en charge notamment de la collecte des impôts et taxes et de la législation fiscale.

Cette administration possède une hiérarchie forte séparée en 8 services, qui définit leur domaine de travail, ainsi que différentes directions (voir organigramme 1 en annexes). Une majorité des services sont des services métiers, directement en lien avec les missions de la DGFIP, mais 3 de ces services sont des services support, ou *transverses*. Ces derniers sont le service des Ressources Humaines, le service de Stratégie, Pilotage et Budget, et le Service des systèmes d'Information (SI). Malgré une interaction indirecte avec le domaine métier des finances publiques, ils répondent aux besoins de la DGFIP en permettant le bon fonctionnement des autres services, ou même en améliorant leur performance.

Comme toute grande structure aujourd'hui, la DGFIP possède un besoin très fort en technologies de l'information, qui est rempli par le SI. Ce service est indispensable, car de nombreuses missions de la DGFIP reposent sur des programmes (e.g., calcul et déclarations d'impôts et des taxes) permettant de traiter de larges quantités de données en un temps restreint. Les premières versions de ces applications datent des années 80, et ont pour la plupart évolué et sont restées utilisées jusqu'à aujourd'hui. L'informatique est donc au centre de cette administration, autant pour les agents en interne que pour les utilisateurs externes.

Le SI est séparé en de nombreux bureaux (voir annexe 2). Il comprend lui-même des bureaux *transverses* qui facilitent le bon fonctionnement des autres bureaux. Le reste des bureaux est regroupé sous la Direction des projets numériques (DPN), et sont chargés du pilotage, du développement et de la maintenance d'applications d'un domaine précis.

CHAPITRE 1. LA DGFIP, ADMINISTRATION EN ÉVOLUTION CONSTANTE

Cette nouvelle hiérarchie date de 2021, où une réorganisation a eu lieu. En effet, la DPN n'existait pas avant cela, et les bureaux étaient regroupés sous deux directions, "étude et développement" et "production". Le SI comporte aujourd'hui plus de bureaux, qui sont donc plus spécialisés, avec par exemple des bureaux en charge d'une unique mission importante.

Le Bureau du SI des professionnels (BSI-3) est un bureau de la DPN chargé de la fiscalité des professionnels, dirigé par Alain Kerdoncuff. Il a à sa charge une dizaine d'applications qu'il spécifie, développe et maintient. L'une d'entre elles est MEcanisation Des Opérations Comptables (MEDOC) qui est une application d'encaissement d'impôts et de gestion de comptabilité de l'État. Le BSI-3 possède une mission particulière de modernisation de cette application, tâche très complexe étant donné son échelle et sa complexité.

Ce bureau était auparavant nommé SI-1C, et ses missions n'ont pas changé avec la réorganisation. Cependant, chacun des bureaux sont encore spécialisés en plusieurs divisions, chacune en charge de projets spécifiques. Ces divisions ont-elles changé avec la réorganisation, notamment une en particulier qui a été supprimée, la Division technique transverse (DTT). Cette dernière, chapeautée par Olivier Blanc (aussi mon tuteur) était à la fois une aide technique sur le domaine du logiciel, ainsi qu'une division détachée des projets principaux, permettant de mettre en avant des technologies innovantes et des projets expérimentaux.

Parler de la cellule innovation ?

Détailler avec des chiffres

Chapitre 2

Notions techniques

2.1 Que sont les logs ?

Le terme “log” est tiré de l’anglais et signifie a l’origine “journal”, tel un journal de voyage. Il décrit donc un document contenant une liste chronologique d’évènements datés, qui peuvent être utilisés pour reformer l’histoire de ce qui s’est passé. Ils sont utiles dans le cas d’un accident, comme par exemple les journaux décrivant les voyages d’explorateurs ou aujourd’hui les boites noires dans l’aviation.

Les logs informatiques découlent de cette définition, décrivant le chemin d’exécution d’une application, d’un site, ou même d’un système d’exploitation. Ces fichiers de logs sont séparés en lignes de logs (souvent appelée un “log”) qui contiennent des informations communes a chaque ligne, ainsi que le message de log qui est la partie variable. Les informations communes vont en général être au minimum la date, l’heure et le niveau du log. Le niveau représente la gravité de l’information présentée, en général décomposée de cette façon :

0. DEBUG : Log utile pour débbugger l’application, qui sera en général utilisé par le développeur. Ce log ne devrait pas être trouvé dans des fichiers de logs d’un logiciel en production.
1. INFO : Information sur le statut fonctionnel d’une application, qui peuvent être utiles et compréhensibles d’un point de vue métier.
2. WARNING : Anomalie logicielle non bloquante pour l’application, peut être fonctionnelle ou logicielle mais ne nécessite pas d’action immédiate.
3. ERROR : Anomalie logicielle bloquante, souvent un bloc logiciel qui cesse de fonctionner. Nécessite en général une intervention assez rapide, pour éviter d’autres erreurs au sein du système.

Grâce a ces informations, un développeur (ou quelqu’un chargé de la maintenance) sera capable de retrouver les anomalies qui ont eu lieu ainsi que de remonter les causes de ces anoma-

lies. C'est un processus en général assez manuel, qui nécessite souvent de comparer les logs et le code source, et où la recherche par mot clé peut s'avérer très utile.

Ajouter une section sur les réseaux de neurones ?

2.2 Vectorisation sémantique

La vectorisation sémantique est un processus de traitement de texte permettant de traiter une chaîne de mots et d'en tirer un "vecteur sémantique", qui encode la signification de cette phrase sous forme d'un certain nombre de valeurs numériques. Si ces vecteurs ne sont pas très utiles à eux seuls, ils sont très utiles pour les comparer entre eux. Voici des exemples communs de ce qui est possible, en notant $v(\text{mot})$ le vecteur sémantique d'un mot :

$$\begin{aligned} v(\text{homme}) - v(\text{femme}) &\simeq v(\text{roi}) - v(\text{reine}) \\ v(\text{homme}) - v(\text{femme}) + v(\text{fille}) &\simeq v(\text{garçon}) \end{aligned} \tag{2.1}$$

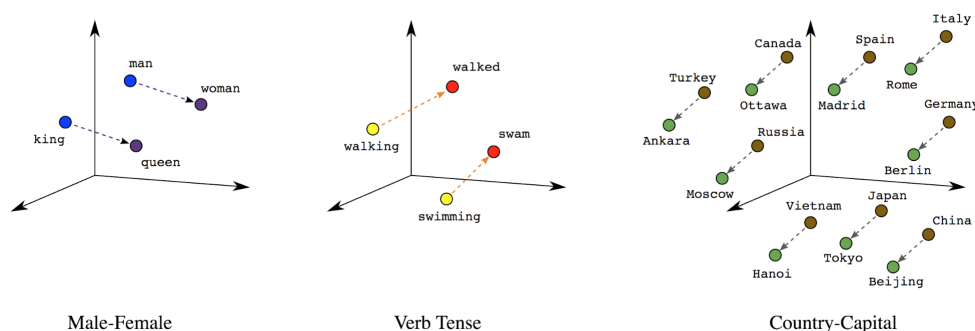


FIGURE 2.1 – Exemple de relations entre vecteurs sémantiques

Ces vecteurs sont générés par un processus relativement simple quand on connaît les bases de l'apprentissage automatique. Je vais brièvement expliquer ici deux méthodes, présentées en plus de détail dans le papier word2vec [7]. La figure 2.2 décrit les structures des modèles présentés, qui sont en fait symétriques l'un de l'autre.

Chacune de ces méthodes utilise les vecteurs sémantiques comme entrée et sortie du réseau, et au début de l'entraînement chaque mot est lié à un vecteur aléatoire. En utilisant un corpus de texte, un mot ainsi que les mots qui l'entourent sont utilisés pour améliorer les valeurs du vecteur du mot central. Pour le CBOW (Continuous Bag Of Words, ou sac de mots continu), le "contexte" autour du mot est donné, et on entraîne le réseau à deviner le mot actuel. À l'inverse, pour le modèle skip gram, on donne le mot initial et on fait deviner les mots qui l'entourent.

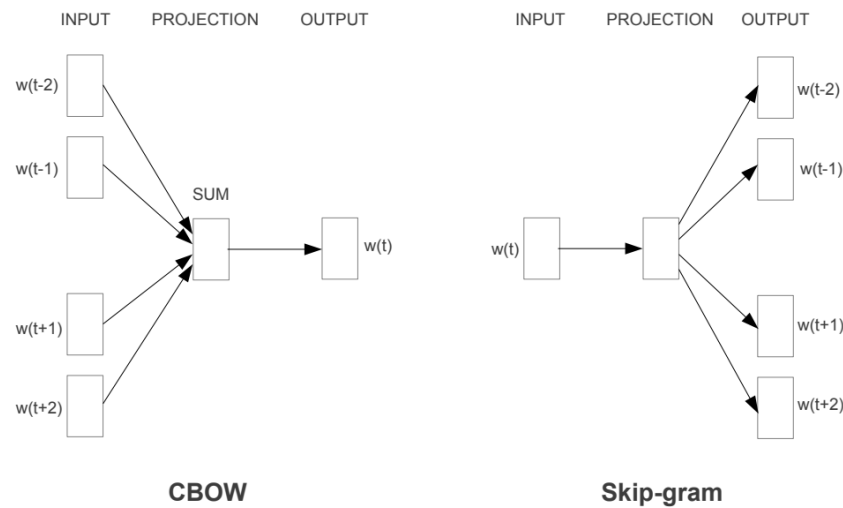


FIGURE 2.2 – Architectures de vectorisation de word2vec

Cette méthode présente des résultats très concluants, et de nombreuses méthodes plus récentes s'en inspirent pour résoudre des problèmes similaires. Pour les utiliser en pratique, il n'est pas réellement nécessaire de refaire toute cette procédure d'entraînement. Il suffit de récupérer une sorte de dictionnaire, qui affecte à chaque mot son vecteur sémantique. Il existe de ces dictionnaires pour beaucoup de langages, et notamment en français. Il est important de noter que certains contextes nécessiteraient de ré-entraîner un modèle, mais c'est très rare.

Tester fonctionnement [Polytechnique NLP](#)

2.3 Forêt d'isolation

La forêt d'isolation, décrite en 2009 [6], est une méthode d'apprentissage automatique classique utilisée pour la détection de valeurs aberrantes. Elle fonctionne en déterminant un coefficient d'isolation pour chaque valeur d'un jeu de données, déterminée par la facilité à la séparer du reste des données.

L'algorithme est exécuté de cette façon :

1. On choisit aléatoirement une variable (exemple, x ou y en 2 dimensions)
2. On choisit une valeur de seuil aléatoire, entre le maximum et le minimum de cette variable dans les points non isolés
3. On sépare les points inférieurs et supérieurs à ce seuil
4. Si un point de donnée se retrouve isolé du reste, on ne le compte plus pour l'algorithme, et on mémorise le nombre de "splits" qu'il a fallu faire pour l'isoler

5. Si tous les points n'ont pas été isolés, on retourne à l'étape 1.

A la fin de cette boucle, un score d'isolation compris entre 0 (valeur normale) et 1 (anomalie) est calculé en fonction du nombre de splits. L'algorithme utilisant de l'aléatoire, il est possible de l'exécuter plusieurs fois pour obtenir une valeur moyenne d'isolation pour chaque point. Un exemple de séparation pour un point normal et aberrant est montré en figure 2.3.

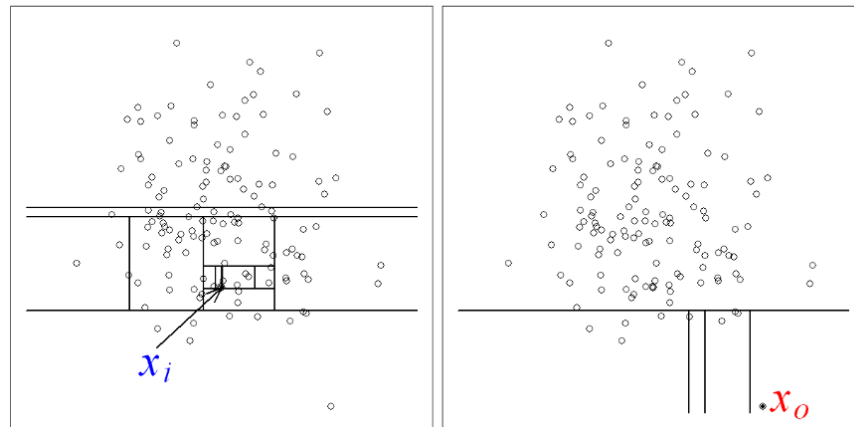


FIGURE 2.3 – Fonctionnement des forêts d'isolation

Chapitre 3

L'existant : le projet analyse-logs

3.1 Contexte du projet

Si le premier chapitre peut mettre quoi que ce soit en avant, c'est bien le besoin très fort en informatique de la [DGFIP](#), rempli par le [SI](#) et ses très nombreuses applications. Mais un parc applicatif si étendu pose un problème majeur : celui de la maintenance. En effet, la maintenance logicielle est un processus très coûteux, surtout en termes d'heures de travail de personnes qualifiées. Ce sont les développeurs des applications à maintenir qui doivent effectuer cette maintenance, car c'est eux qui connaissent le fonctionnement interne de l'application.

Une solution serait de former des équipes de maintenance au différents logiciels. Malheureusement, cela demanderait encore plus de moyens, à la fois prenant encore du temps aux développeurs, mais aussi nécessitant d'embaucher des personnes qui travailleraient à temps plein sur la maintenance, ce qui n'est pas envisageable. Il n'est même pas donné que cela libère vraiment du temps aux développeurs, étant donné l'évolution constante des logiciels et les formations supplémentaires qu'il faudrait donner pour tenir une équipe de maintenance à jour.

Il y a 4 ans, mon tuteur Olivier Blanc s'est penché sur la question. Le vrai problème était bien de faire gagner du temps aux développeurs en facilitant et accélérant la maintenance d'une application. Dans le cas d'une erreur dans l'application qui stopperait partiellement ou complètement son fonctionnement, sa correction est obligatoire pour les équipes. Le problème de retrouver la source de cette erreur et de la corriger est alors aussi important, et c'est ici que rentrent en jeu les [logs](#) applicatifs (voir partie suivante). Olivier mit en avant le papier de recherche DeepLog [3], dont le but était d'utiliser les logs pour détecter et prédire les erreurs, ainsi que remonter vers leur cause initiale. Le projet analyse-logs est alors né, commençant comme sujet de PFE de Rémi Grison, apprenti à la DGFIP en 2019. Léa Lebert a ensuite succédé à Rémi, puis un an après le départ de Léa j'ai repris le projet.

3.2 Le modèle original : DeepLog

Le travail de Rémi qui a démarré ce projet a principalement été l'implémentation de l'algorithme DeepLog à partir du papier de Du et al. [3]. Je vais ici détailler le fonctionnement de DeepLog ainsi que l'implémentation qui a été faite par Rémi.

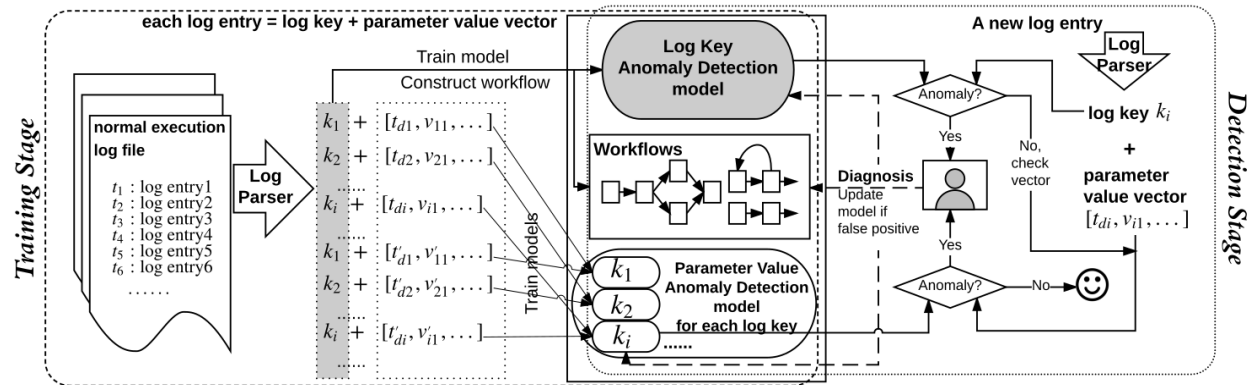


FIGURE 3.1 – Structure de DeepLog

La structure complète du modèle est détaillée dans la figure 3.1, tirée directement du papier. Une étape qui n'est pas détaillée ici est celle de la récupération et du regroupement des logs, qui n'est pas aussi triviale qu'elle pourrait en avoir l'air. Cependant, ce point ne sera pas plus approfondi dans ce rapport étant donné que nous ne l'avons pas vraiment traité.

Parsing

La première étape est celle du "parsing" (que l'on pourrait approximativement traduire par analyse syntaxique en français). Elle est appliquée à un fichier de logs pour le traduire en une liste d'ids et de vecteurs de paramètres. Par exemple, avec les logs suivant :

```
Session ouverte id 123
Session ouverte id 456
Session fermée id 123
Session fermée id 456
```

On s'attend à obtenir un tel résultat parsé :

```
1, [123]
1, [456]
2, [123]
2, [456]
```

La méthode utilisée par DeepLog pour produire ces résultats est Spell [2], développée l'année précédente par une partie des chercheurs qui ont travaillé sur DeepLog. Seulement, entre la

publication du papier et l'implémentation par Rémi, 2 années sont passées, et l'état de l'art avait changé en faveur de l'algorithme Drain [5]. Cet algorithme avait été publié la même année que DeepLog, et est encore aujourd'hui une solution de choix pour le parsing de **logs**. Rémi a donc produit sa propre implémentation de l'algorithme Drain pour le projet. Il existe aujourd'hui une très bonne implémentation open source de drain en Python, qui est donc plus robuste et performante que l'implémentation de Rémi, et nous allons donc la réutiliser plus tard.

Analyse des suites

L'étape d'analyse de DeepLog est séparée en deux parties. Les chercheurs de DeepLog avaient pour but de créer un modèle plus puissant que ce qui a pu exister auparavant. Un des problèmes de ces anciens modèles est l'absence de prise en compte de la temporalité, les **logs** étant seulement analysés un par un sans prendre en compte leur ordre d'apparition. L'analyse de l'enchaînement des **logs** est donc très importante pour mieux comprendre certaines erreurs moins évidentes à premier abord.

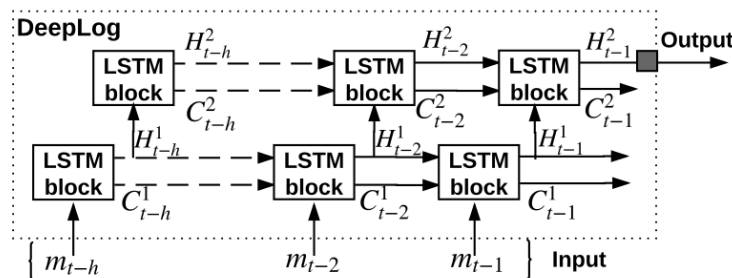


FIGURE 3.2 – Structure d'un réseau LSTM empilé

L'analyse temporelle est faite par DeepLog à l'aide de réseaux **LSTM** empilés (voir figure 3.2). Un **LSTM** est un type de réseau de neurones en **apprentissage profond**, qui possède des connexions de rétroaction qui lui permettent de capturer des comportements sur une durée relativement longue.

Les ids des **logs** sont utilisés, et on entraîne le modèle à prédire les g **logs** les plus probables qui apparaîtraient après une suite de h **logs**, à l'aide d'une fenêtre glissante. Par exemple, pour cet enchaînement d'ids : 1, 2, 1, 3, 4, 1, 2, avec une taille de fenêtre $h = 5$, on va envoyer 1, 2, 1, 3, 4 au réseau, et on s'attend à voir 1 au sein des g **logs** prédits. Ensuite on enverrait 2, 1, 3, 4, 1 en attendant 2, etc.

Cet entraînement pourra ensuite être utilisé durant l'analyse d'un fichier de **logs**, ou même durant son exécution. On prévoit les g ids les plus probables, puis on compare avec le **log** qui est vraiment à la suite. Si son id n'est pas dans les prédictions de l'algorithme, on considère ce **log** comme anormal.

Analyse des paramètres

Le choix des créateurs de DeepLog a été de considérer chaque partie variable des **logs** (soit chaque paramètre) comme sa propre série temporelle à travers les **logs** de même id. En faisant cela, il est possible de réutiliser la même architecture de **LSTM** empilés détaillés à la figure 3.2. Cette fois, on attend une unique prédiction de valeur, et les lignes sont considérées comme anormales si la valeur observée est trop différente de la valeur réelle.

3.3 Evolutions du projet par Léa

Suite à l'implémentation de DeepLog par Rémi, le projet était certes fonctionnel mais très peu utilisable par un utilisateur lambda. En effet, l'utilisation se faisait à travers une ligne de commande comportant de nombreux paramètres, celle-ci étant la seule interface pour les utilisateurs.

Le travail de Léa a donc été d'améliorer ce projet pour le rendre plus simple d'utilisation, mais aussi d'agrandir l'éventail de possibilités d'analyse proposé. Il a été choisi de créer une nouvelle façon de faire l'interface avec le modèle via la bibliothèque Blockly créée par Google, détaillée à la suite. Je ferai aussi une explication des autres algorithmes mis en place, sur quoi j'avais aidé Léa au moment de son PFE.

Blockly

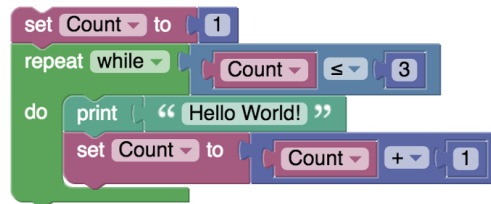


FIGURE 3.3 – Exemple de programme avec Blockly

Blockly est une librairie publiée par Google, permettant de créer des éditeurs de code visuels. L'utilisateur relie des blocs qui représentent des instructions, des variables, des boucles (voir figure 3.3) et cette représentation est traduite en code, de façon entièrement configurable par un développeur. Si vous avez déjà eu affaire au moteur de jeu en ligne Scratch, son éditeur visuel est basé sur Blockly.

Olivier et Léa ont fait le choix d'utiliser cette librairie pour représenter les différentes structures de modèles, ou "pipelines". En effet, si on considère chaque bloc du modèle, il est possible de les remplacer par d'autres blocs équivalents, par exemple un autre analyseur de modèle. Blockly permet très bien cette représentation, avec des couleurs et des formes qui correspondent aux

types des éléments. Chaque étape du modèle pourrait alors avoir son type de bloc, et l'interface serait visuellement très intuitive.

L'interface Blockly a donc été développée par dessus le travail de Rémi, mais aussi avec les autres algorithmes détaillés dans la partie suivante. Elle n'aurait eu que peu d'intérêt si l'idée d'algorithmes/modèles alternatifs n'avait pas été émise, mais cette idée permet de créer facilement et intuitivement des modèles adaptés aux besoins de différentes applications de la DGFIP.



Algorithmes alternatifs

Il existe de nombreux algorithmes qui, à partir de suites d'ids ou de paramètres, seraient capables de prévoir la validité d'une occurrence suivante. En effet, une partie de la force de DeepLog repose dans son architecture très modulaire, et il serait donc envisageable de remplacer une étape par une autre similaire.

Le premier algorithme qui a été choisi est les "forêts d'isolation" [6], qui se base sur une valeur d'isolation d'un point de données : plus il est facile de le séparer du reste des données, plus il a de chances d'être une anomalie (voir section 2.3). Un module d'analyse de paramètre a été développé avec une implémentation des forêts d'isolation, capable de remplacer celui de DeepLog dans l'implémentation de Rémi.

C'est ici que j'ai eu mon premier contact avec le projet durant ma première année d'alternance. Il m'a fallu assister Léa sur la fin de son PFE, pour effectuer des tâches qu'elle n'avait pas le temps de faire. J'ai aussi pu apporter une aide technique sur la partie [apprentissage automatique](#) du projet, étant suite à avoir suivi un cours à ce sujet.

En m'inspirant de ce qui a été fait sur les forêts d'isolation, j'ai fait le choix de créer un module d'analyse de suites et de paramètres générique basé sur la librairie d'[apprentissage automatique](#) scikit learn. C'est une bibliothèque libre présentant de nombreux algorithmes communs d'[apprentissage automatique](#), tels que les forêts d'isolation. Utiliser une librairie telle que scikit me permet tout d'abord une meilleure robustesse et performance des algorithmes, mais aussi une certaine généralité. En effet, les algorithmes du même type ont tous la même structure de fonctions, et cela permet de très facilement les interchanger.

Les modules qui ont été développés permettent donc de remplacer l'analyse de paramètres de DeepLog par n'importe quel algorithme de la bibliothèque scikit, s'intégrant parfaitement dans la structure par blocs proposée par Blockly. Trois algorithmes de détections d'anomalies ont été retenus : les forêts d'isolation (version scikit), les SVM (Support Vector Model, ou [Séparateur à Vaste Marge](#) ) à une classe, et les LOF ([Local Outlier Factor](#) ). Ce sont trois méthodes de l'état de l'art en apprentissage non profond pour la détection d'anomalie, qui pourraient s'avérer utiles pour l'analyse de paramètres en particulier.

Chapitre 4

Le processus scientifique au cœur du projet

Nous arrivons maintenant au cœur de ce mémoire, qui démarre au début de mon PFE, et plus précisément l'arrivée de Robin à la [DGFIP](#). Robin était étudiant en Master ALMA à l'université de Nantes, et a travaillé avec moi sur le projet d'analyse de [logs](#) en tant que sujet de stage de fin de master. Il s'est chargé de beaucoup de travail de documentation sur le sujet ainsi que des sujets plus axés recherche.

Je vais détailler au sein de ce chapitre cette grande partie du projet et les différents points important que nous avons rencontrés.

4.1 État de l'art de l'analyse de logs

À l'arrivée de Robin au bureau, j'étais encore en période école. Il a donc commencé le travail sur le projet avant moi, et ce en commençant la rédaction d'un état de l'art des algorithmes d'analyse de [logs](#) existants. Ce travail est un travail de longue haleine, la compréhension des papiers n'étant pas du tout triviale. Malgré les meilleurs efforts des chercheurs les ayant rédigés, un long travail de documentation et d'apprentissage est nécessaire pour comprendre certains des concepts abordés et utilisés.

Un papier qui a beaucoup aidé cette recherche documentaire est le rapport d'expérience [1] publié par Huawei et l'université de Hong Kong en 2021. Ce document étant très récent, étant publié environ 6 mois avant le début du projet, il nous a semblé être une source fiable d'informations et de données au sujet de la détection d'anomalies dans les logs. Il compare notamment DeepLog à cinq autres méthodes, pour un total de trois méthodes (semi-)supervisées et trois non-supervisées.

Nous avons eu beaucoup d'échanges avec Robin durant ses recherches documentaires, au sujet des algorithmes trouvés, de détails incompris ou de points intéressants. Nous avons pu

comme cela m'aide à mieux comprendre les différents papiers trouvés, et faire des choix et des hypothèses au sujet de l'implémentation et des résultats.

Cette recherche nous a mené à choisir les algorithmes que nous souhaitons implémenter pour la suite, que je vais détailler dans la suite de cette section. Nous avons choisi ces algorithmes en fonction de notre compréhension des mécanismes en jeu, de leur facilité d'implémentation perçue, ainsi que de leur utilité vis à vis de notre problème.

LogRobust

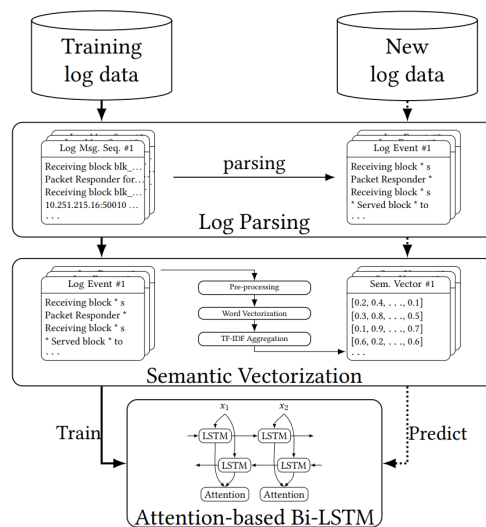


FIGURE 4.1 – Structure de LogRobust

LogRobust [8] est un modèle d'analyse de logs qui montre des similarités avec DeepLog, publié en 2019. Comme son nom l'indique, l'objectif était de pallier aux problèmes des anciennes méthodes de l'état de l'art en proposant une solution robuste. L'architecture du modèle est détaillée sur la figure 4.1.

La plus grosse différence de ce modèle comparé à DeepLog est l'étape de vectorisation sémantique présente à la suite du parsing. Si DeepLog utilise les id de logs et les valeurs des différents paramètres pour reconnaître les anomalies, LogRobust utilise la sémantique des logs et leur enchaînement (voir section 2.2 pour plus de détails). Cette analyse sémantique permet effectivement de lisser les valeurs utilisées, d'où la promesse de robustesse de l'algorithme. Des lignes ayant la même signification, par exemple un log dans lequel des synonymes ont été remplacés dans une mise à jour, resteront proche, et ne nécessiteront pas un ré-entraînement du modèle.

Une autre différence notable comparé à DeepLog est l'utilisation d'un bi-LSTM avec un mécanisme d'attention (voir figure 4.2). Le fonctionnement d'un bi-LSTM est similaire au LSTM utilisé

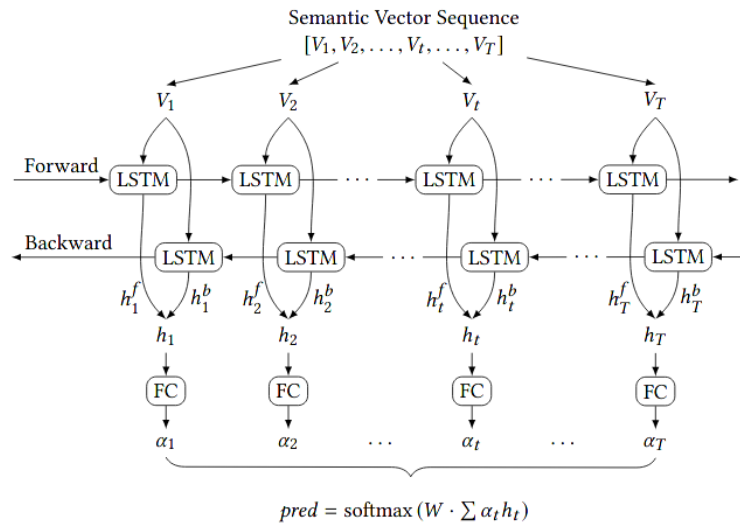


FIGURE 4.2 – Structure d'un LSTM bidirectionnel

par DeepLog, mais on en utilise un dans le sens chronologique (comme DeepLog) et un dans le sens inverse, avec des entrées allant du futur vers le passé. Cela permet de rajouter un contexte temporel aux **logs**, ce qui améliore en général les performances. Un problème posé par cette approche est la nécessité de connaître un certain nombre d'entrées dans le futur, et dans le cas d'une analyse en direct un délai supplémentaire est alors nécessaire.

Le mécanisme d'attention simule le concept d'attention humaine. Par exemple, quand on lit une phrase, on est capable de savoir quel mots lus précédemment sont important relativement au mot que l'on lit maintenant. C'est ce que fait LogRobust avec les **logs**, en apprenant au fur et à mesure quel **logs** de l'historique sont importants vis à vis du log analysé.

La robustesse du modèle est testée en modifiant les fichiers de logs de différentes façons, de manière aléatoire :

- En modifiant les clés des **logs** :
 - Ajout de mots et/ou de paramètres
 - Suppression de mots
- En modifiant l'enchaînement des **logs** :
 - Suppression de **logs**
 - Mélange de **logs**
 - Duplication de **logs**

Ces fichiers de **logs** modifiés sont ensuite utilisés par les chercheurs pour évaluer la robustesse du modèle LogRobust. Comparé à des méthodes d'apprentissage automatique classiques (non

profond), les performances sont très concluantes. En effet, la méthode performe bien face à des niveaux d'injection jusqu'à 20%, et l'attention est mise en valeur car elle permet de garder les résultats corrects à des niveaux d'injection élevés.

Cette méthode serait donc une très bonne méthode à implémenter pour être durable dans le temps. Il n'est pas nécessaire de la ré-entraîner fréquemment (ce qui pouvait être un problème avec DeepLog). Il est important de noter que cette méthode ne prend pas en compte les valeurs des paramètres des `logs`, ce qui pourrait être un problème dans certains cas d'application. Mais la robustesse de la méthode liée à son mécanisme d'attention, ainsi que son architecture relativement simple, sont de bons arguments pour son utilisation.

AutoEncoder

Cette méthode publiée en 2020 [4] est appelée "AutoEncoder" dans le reste du document, mais elle utilise en réalité 2 réseaux de type autoencoder ainsi que des forêts d'isolation.

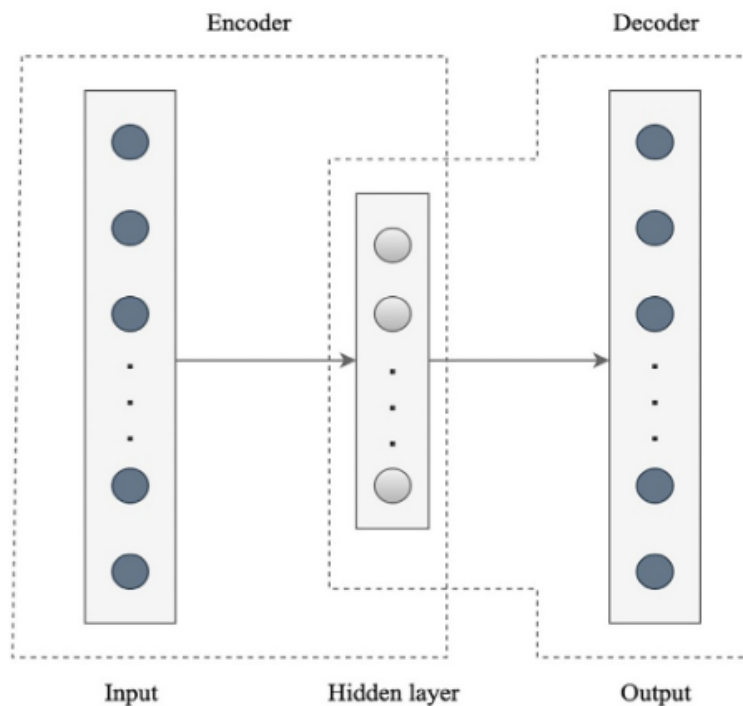


FIGURE 4.3 – Structure d'un réseau autoencodeur

Le plus important pour comprendre ce modèle est le fonctionnement d'un AutoEncoder. C'est encore une fois un réseau de neurone, avec une structure simple en "double entonnoir" (voir figure 4.3). L'objectif de cette structure est d'apprendre un encodage (et décodage) de données pour réduire la taille de cette donnée, avec une perte d'information minimale. La valeur encodée est

donc ce que l'on va retrouver dans la couche cachée au centre du réseau, et notre apprentissage nous permet d'encoder et de décoder des données par la suite.

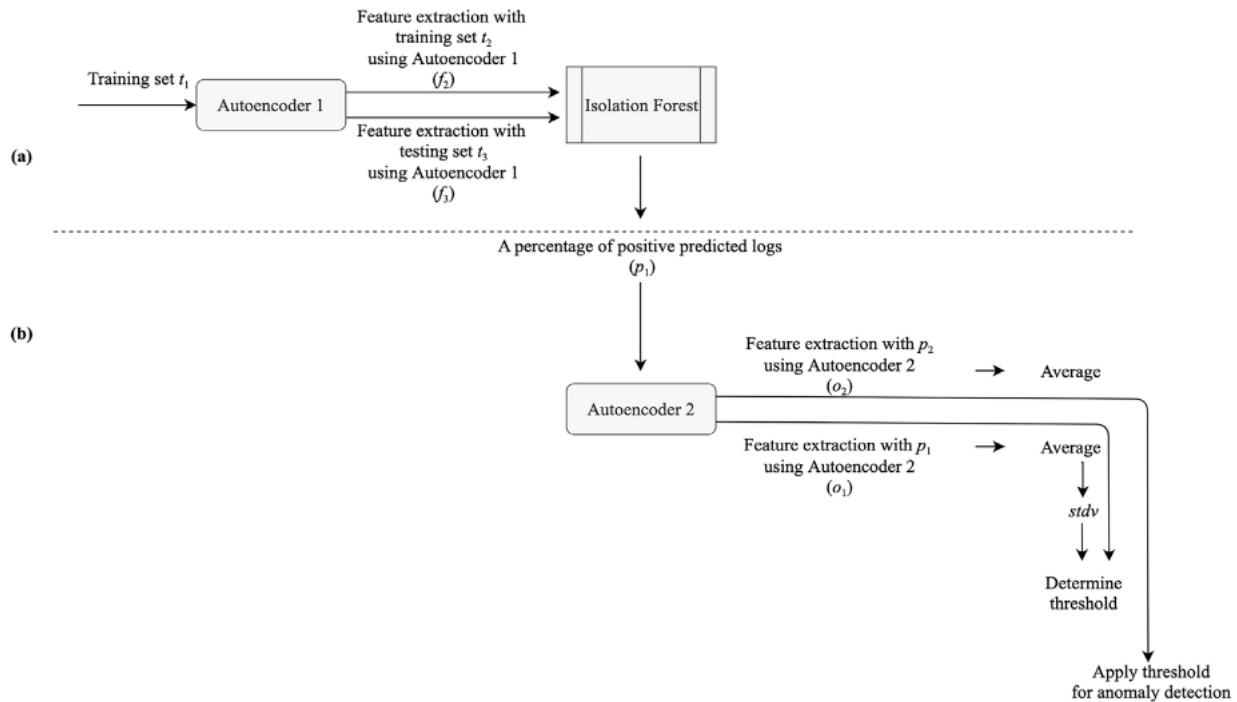


FIGURE 4.4 – Structure du modèle “AutoEncoder”

La structure complète du modèle est détaillée dans la figure 4.4. Le premier autoencoder est entraîné à encoder les **logs**, soit en extraire les variables significatives. Ensuite, la forêt d'isolation fait une première passe de détection d'anomalies selon ces **logs** encodés, ce qui à la fois accélère le processus et améliore la détection. Enfin, une seconde passe de détection d'anomalies est faite avec un second autoencoder, qui va récupérer à la fois les valeurs négatives détectées par la forêt d'isolation et les valeurs positives. Cette passe permet de déterminer et utiliser un seuil d'acceptation des **logs**.

C'est une structure très complexe, et je ne saurais décrire en détail les raisons de sa structure si particulière. Mais c'est une méthode qui n'est pas trop compliquée à mettre en place malgré tout, se basant sur des concepts bien connus et implémentés dans des bibliothèques, qu'il suffit de composer. Les résultats détaillés dans le papier sont assez concluants, et les comparaisons du rapport d'expérience [1] montrent qu'il est notamment très robuste pour des jeux de données avec beaucoup d'anomalies, ce qui n'est pas le cas des autres modèles.

4.2 Analyse des jeux de données disponibles

Nous avons réussi à obtenir un large jeu de données de **logs** de l'application **MEDOC** dont j'ai parlé succinctement dans la section 1. Elle produit une grosse quantité de **logs** liés notamment aux parcours utilisateurs durant leur sessions sur l'application. Voici un exemple de **log** tiré de ce jeu de données :

```
2021-05-14 07:35:17,106 - http-bio-8140-exec-32 - INFO -  
| fr.gouv.finances.medoc.persistance.cpl.dao.VerrouDao.  
| leverVerrousUtilisateur(VerrouDao.java:167) - 0 Supprimés pour  
| la session SERV0070100-AG323056-0DB4EE01F8D31E5C4D2EE3BDA6131B62
```

On retrouve ici les différents éléments abordés dans la section 2.1, tels que la date, l'heure et le niveau du **log**. Seulement, on retrouve aussi des informations supplémentaires telles que le fil d'exécution, la classe et la méthode d'origine. Ce sont des informations supplémentaires qui peuvent être très utiles à un développeur pour tracer l'origine d'une erreur.

De nombreux **logs** comportent aussi un identifiant de session, par exemple le long identifiant lisible à la fin de la ligne d'exemple ci-dessus. La gestion des sessions est un point très important dans l'analyse de **logs**, en particulier quand on prend en compte l'enchaînement des **logs** comme le font DeepLog et LogRobust. En effet, si les sessions se retrouvent entrelacées dans les **logs**, les enchaînements peuvent devenir imprévisibles, et une anomalie peut être détectée là où il n'y en a pas.

Il existe plusieurs solutions à ce problème d'entrelacement :

- Désentrelacer les **logs** : Si tous les **logs** appartenant à des sessions différentes contiennent un identifiant de sessions et que son extraction est possible, il est possible de les désentrelacer. Il serait alors envisageable de traiter les sessions une par une, ou bien de les regrouper au sein même du flux des **logs**.

Cette solution pose certains problèmes qui nécessiterait des changements de structure des modèles, notamment pour la gestion de successions des **logs** avec des heures qui ne se suivent pas.

- Augmenter la taille de l'historique : L'analyse se fait avec un historique de **logs** d'une taille fixe, à partir du quel on prévoit le log suivant. Si cet historique est grand, précisément plus grand que la taille totale des sessions entrelacées, l'algorithme serait capable de ne prendre en compte que les **logs** intéressants de cet historique. Cela est d'autant plus vrai pour un système implémentant un mécanisme d'attention.

La solution la plus évidente à mettre en place serait donc la seconde. Un point crucial est donc de connaître ces tailles de session dans nos jeux de données de **MEDOC**, et d'autres données tirées de ces **logs** pourraient aussi nous être utiles.

Une analyse des données a donc été mise en place via un notebook Python Jupyter, comme nous avons utilisé cet outil en cours pour ce même genre de tâches. Un notebook permet de séparer du code Python en différents blocs exécutables séquentiellement et séparément, ce qui est utile pour du traitement de données et des visualisations.

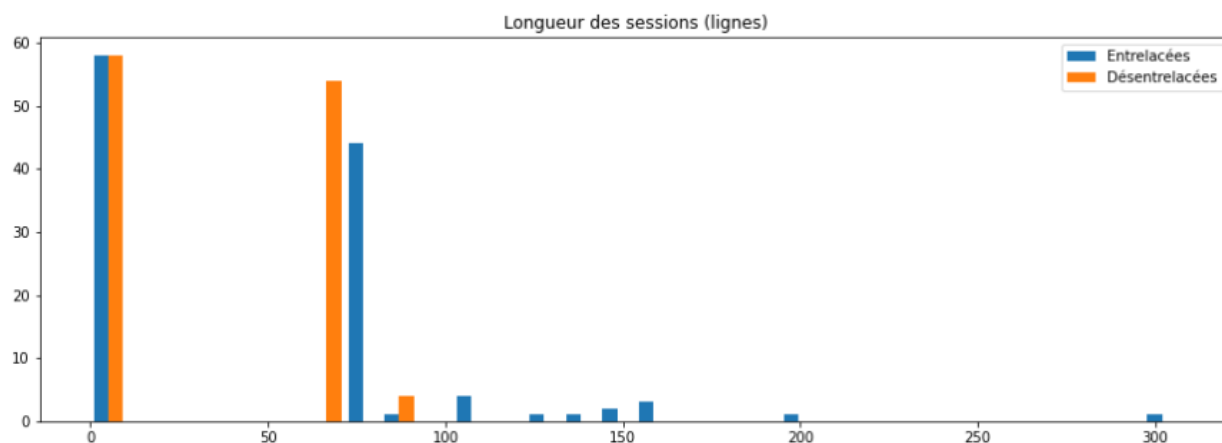


FIGURE 4.5 – Longueur des sessions dans les logs MEDOC

La première donnée sortie de cette analyse est la longueur des sessions, entrelacées ou non pour permettre une comparaison (figure 4.5). Cette première analyse amène déjà beaucoup d'informations sur le comportement des sessions. D'abord, on observe de nombreuses sessions contenant un unique `log`, ce qui est un peu suspect et qu'il faudra étudier par la suite. Ensuite, l'entrelacement des sessions est assez minime mais cause quand même une explosion de la longueur maximale de session : une session passe ici de moins de 100 lignes à elle seule, à 300 lignes quand entrelacée avec le reste des `log`. Enfin, on observe aussi l'existence d'une session qui comporte plus de 1000 lignes quand désentrelacée.

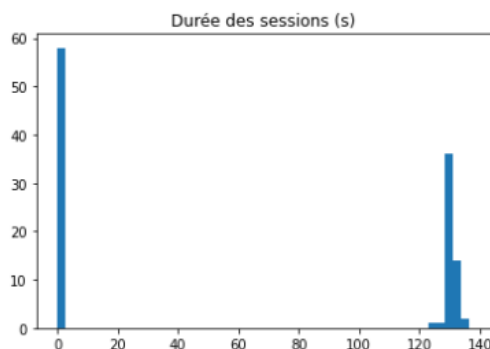


FIGURE 4.6 – Durée des sessions

Une autre analyse qui a été effectuée est celle de la durée dans le temps des sessions (voir

figure 4.6). On retrouve ici les sessions qui ne contiennent qu'un unique `log`, et ont une durée nulle. Sinon, on peut observer que les sessions ont une durée très proche, autour de 2 minutes 10 secondes.

Une analyse plus détaillée des sessions de 1 ligne nous indique qu'elles sont toute, sans exceptions, des sessions "Catalina". En effet, les autres sessions comportent toutes au minimum un message d'ouverture et de fermeture de session, et des traitements sont toujours effectués entre ces deux événements. Mais que sont ces sessions Catalina ? Catalina est un composant de Tomcat, le moteur de serveur utilisé à la DGFiP. Catalina est en fait l'implémentation du moteur de Tomcat, ou plus précisément des "servlets" Tomcat (voir article wikipédia [Servlet](#) pour plus d'informations). Ces `logs` représentent des fermetures de sessions Catalina, qui ne sont a priori pas les mêmes que les sessions utilisateurs qui nous intéressent.

Tout ce qu'il nous faut savoir, c'est que ces sessions ne sont pas très intéressantes à mettre à part du reste des `logs`. Heureusement, tous ces `logs` ont la même structure :

```
2021-05-14 08:06:16,078 -  
| ContainerBackgroundProcessor[StandardEngine[Catalina]]  
| - INFO - fr.gouv.finances.commun.jee9.mapi.presentation.session.  
| BasicSessionListener.sessionDestroyed(BasicSessionListener.java:92)  
| - Destruction de session '<id>'
```

Ce sont les seuls `logs` originaires du fil d'exécution `ContainerBackgroundProcessor[StandardEngine[Catalina]]`, il est alors possible de s'en servir comme filtre et de retirer ces `logs` de l'analyse des sessions, présenté en figure 4.7.

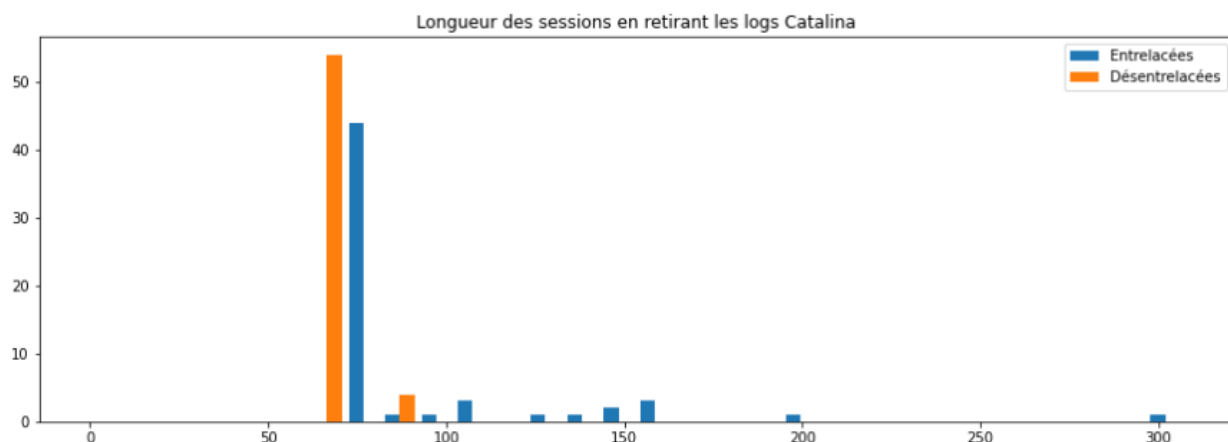


FIGURE 4.7 – Longueur des sessions en retirant les logs Catalina

Une dernière analyse effectuée est celle de la composition des `logs`. Le fichier de `logs` utilisé n'est qu'un extrait de tout le jeu de données auquel nous avons accès, par souci de performance.

Mais cette analyse nous permet quand même de détailler la structure des **logs** avec lesquels nous devons travailler. La figure 4.8 détaille les pourcentages de **logs** appartenant a une session, ainsi qu'une session Catalina.

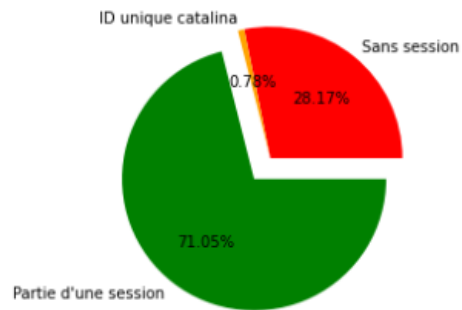


FIGURE 4.8 – Proportions de logs appartenant à des sessions

Chapitre 5

Organisation du projet

Partie orga

5.1 La planification et les aléas

Difficulté de planifier un projet de recherche, évolutions du planning, aléas rencontrés

5.2 Agilité et recherche : une union difficile

Agilité pas mise en place, pourquoi ?

Chapitre 6

Développement du nouveau projet

Technique

Chapitre 7

Projection de l'impact du projet

7.1 Gain de temps et de productivité

Partie éco

7.2 L'humain contre l'algorithme

Partie humaine

Conclusion

Bibliographie

- [1] Zhuangbin Chen, Jinyang Liu, Wenwei Gu, Yuxin Su, and Michael R. Lyu. Experience report : Deep learning-based system log analysis for anomaly detection, 2021.
- [2] Min Du and Feifei Li. Spell : Online streaming parsing of large unstructured system logs. *IEEE Transactions on Knowledge and Data Engineering*, 31(11) :2213–2227, 2019.
- [3] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 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 '17*, page 1285–1298, New York, NY, USA, 2017. Association for Computing Machinery.
- [4] Amir Farzad and T. Aaron Gulliver. Unsupervised log message anomaly detection. *ICT Express*, 6(3) :229–237, 2020.
- [5] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. Drain : An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 33–40, 2017.
- [6] Fei Tony Liu, Kai Ting, and Zhi-Hua Zhou. Isolation forest. In *Eighth IEEE International Conference on Data Mining*, pages 413 – 422, 01 2009.
- [7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.
- [8] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xincheng Yang, Qian Cheng, Ze Li, Junjie Chen, Xiaoting He, Randolph Yao, Jian-Guang Lou, Murali Chintalapati, Furao Shen, and Dongmei Zhang. 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, ESEC/FSE 2019*, page 807–817, New York, NY, USA, 2019. Association for Computing Machinery.

Glossaire administratif

BSI-3 Bureau du SI des professionnels: Auparavant nommé SI-1C, bureau dépendant de la **DPN**, chargé du développement et de la maintenance des applications du domaine de la fiscalité des professionnels.

DGCP Direction Générale des Comptes Publics: Aussi appelée «trésor public», ancienne administration chargée de la gestion des comptes de l'État et du recouvrement des impôts.

DGFIP Direction Générale des Finances Publiques: Service public de l'État rattaché au ministère des finances, chargé des missions de gestion publique et de fiscalité.

DGI Direction Générale des Impôts: Ancienne administration chargée de la liquidation des impôts.

DPN Direction des projets numériques: Direction au sein du **SI** regroupant les différents bureaux en charge de la direction et de la réalisation de projets dans le numérique.

DTT Division technique transverse: Ancienne division du SI-1C (**BSI-3** aujourd'hui) apportant un soutien technique aux autres équipes et proposant des projets d'innovation.

MEDOC MEcanisation Des Opérations Comptables: Application traitant l'encaissement des taxes (telles que la TVA) ainsi que la génération d'écritures comptables pour l'État.

SI Service des systèmes d'Information: Service support de la **DGFIP** chargé de la gestion informatique et du développement d'applications.

transverse Utilisé fréquemment à la **DGFIP** comme synonyme de transversal, dans le sens "recoupant plusieurs disciplines ou secteurs".

Glossaire technique

apprentissage automatique Ou *machine learning* en anglais, ensemble de méthodes visant à développer des algorithmes généraux basés sur l'apprentissage de données, s'opposant à un algorithme explicite classique.

apprentissage profond Branche de l'[apprentissage automatique](#) se basant sur des réseaux de neurones artificiels, comportant plusieurs couches de traitement de données permettant d'extraire des caractéristiques complexes.

log De l'anglais log (journal), sortie d'une application (souvent un fichier) représentant le chemin d'exécution d'une application (voir section [2.1](#)).

LSTM Long Short Term Memory (Longue mémoire à court terme): Type de réseau de neurones en [apprentissage profond](#) capable de retenir de l'information sur une longue durée.

TAL traitement automatique du langage: Ensemble de méthodes visant à analyser et traiter le langage naturel.

Table des figures

2.1	Exemple de relations entre vecteurs sémantiques	5
2.2	Architectures de vectorisation de word2vec	6
2.3	Fonctionnement des forêts d'isolation	7
3.1	Structure de DeepLog	9
3.2	Structure d'un réseau LSTM empilé	10
3.3	Exemple de programme avec Blockly	11
4.1	Structure de LogRobust	14
4.2	Structure d'un LSTM bidirectionnel	15
4.3	Structure d'un réseau autoencodeur	16
4.4	Structure du modèle "AutoEncoder"	17
4.5	Longueur des sessions dans les logs MEDOC	19
4.6	Durée des sessions	19
4.7	Longueur des sessions en retirant les logs Catalina	20
4.8	Proportions de logs appartenant à des sessions	21

Annexes

1	Organigramme des services de la DGFIP	A
2	Organigramme des bureaux du SI	B

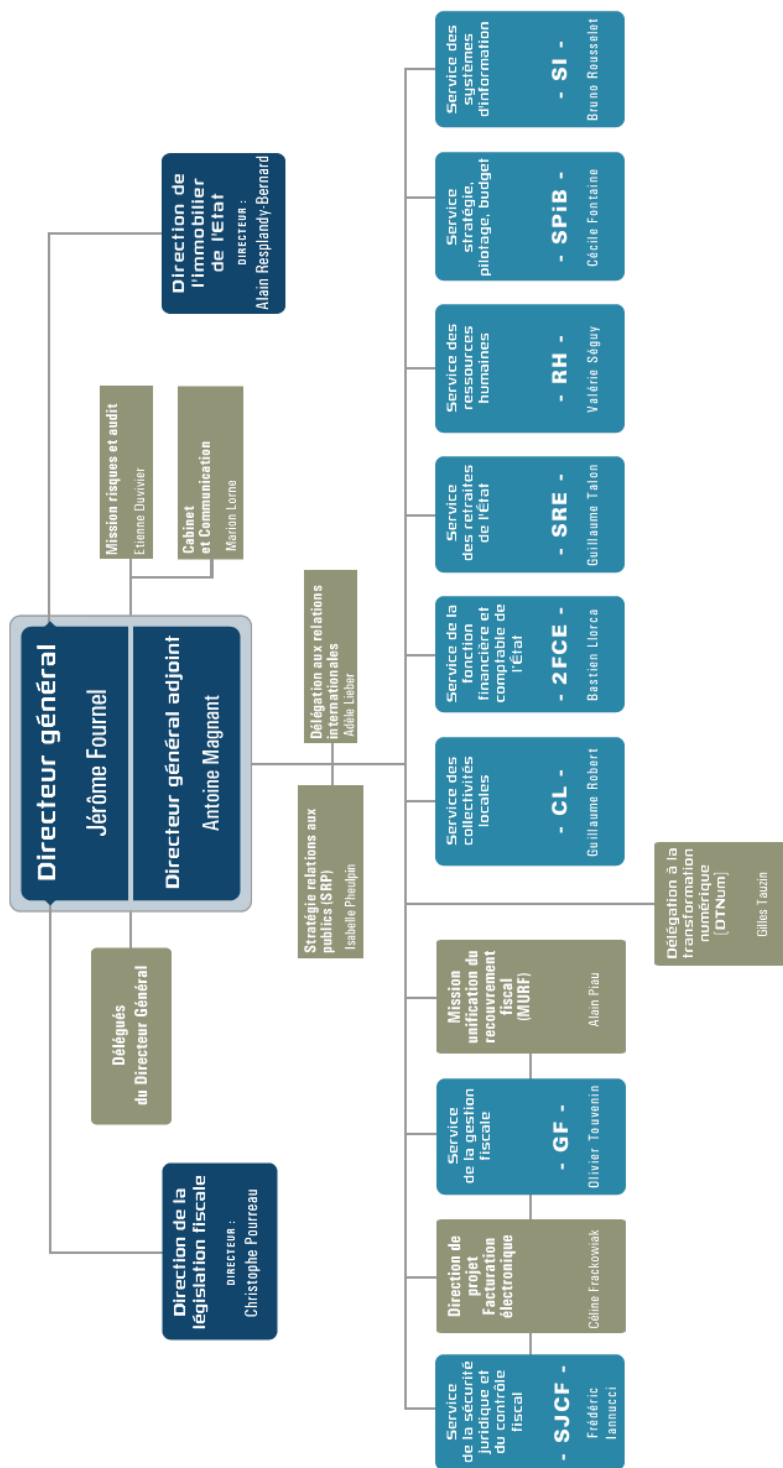


FIGURE 1 – Organigramme des services de la DGFIP

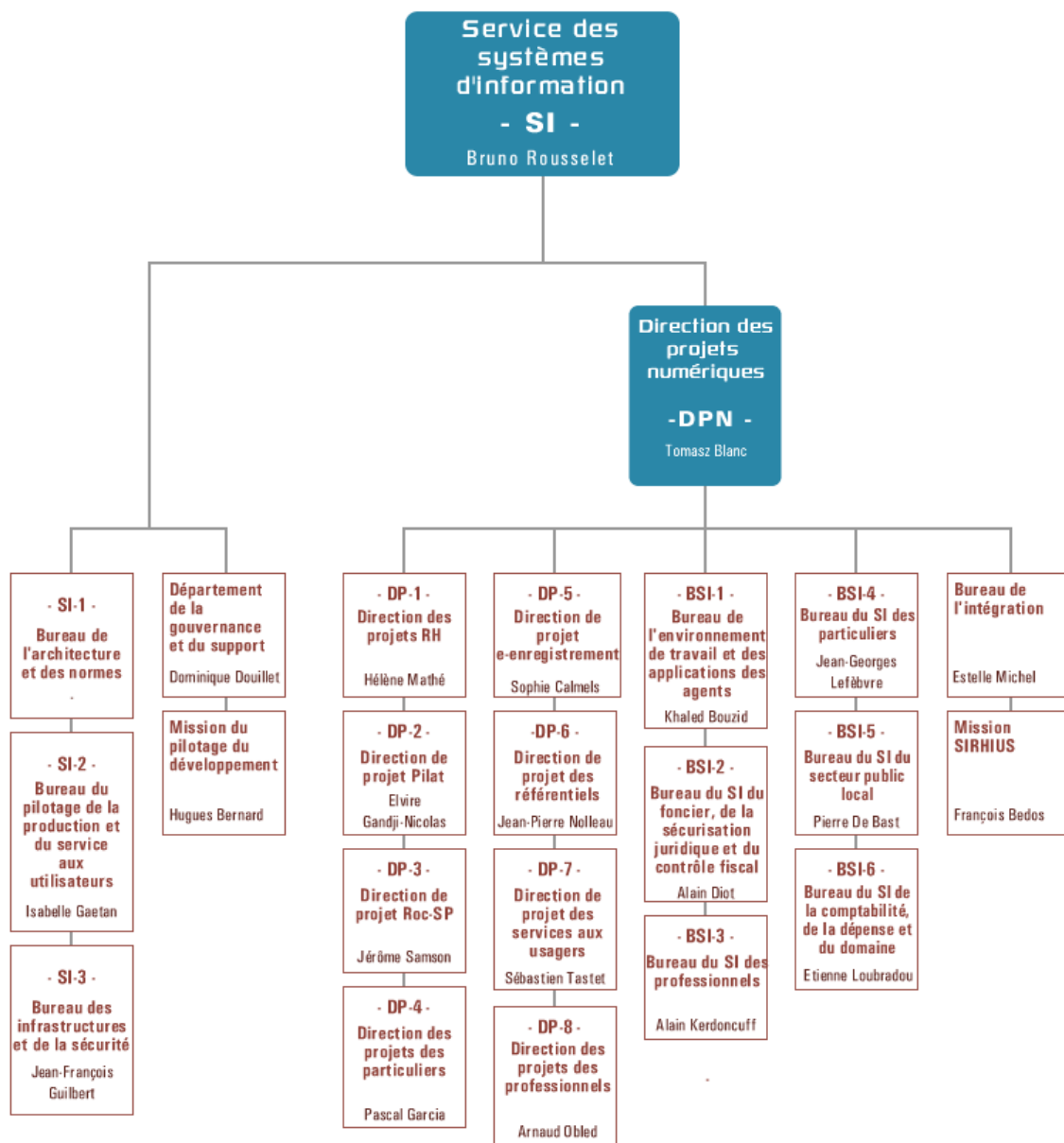


FIGURE 2 – Organigramme des bureaux du SI

Todo list

<input type="checkbox"/> 1-2 pages	1
<input type="checkbox"/> Parler de la cellule innovation ?	3
<input type="checkbox"/> Détailler avec des chiffres	3
<input type="checkbox"/> Ajouter une section sur les réseaux de neurones ?	5
<input type="checkbox"/> Tester fonctionnement Polytechnique NLP	6
<input type="checkbox"/> Partie orga	22
<input type="checkbox"/> Difficulté de planifier un projet de recherche, évolutions du planning, aléas rencontrés	22
<input type="checkbox"/> Agilité pas mise en place, pourquoi ?	22
<input type="checkbox"/> Technique	23
<input type="checkbox"/> Partie éco	24
<input type="checkbox"/> Partie humaine	24