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

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

PAYS DE LA LOIRE

DIRECTION GÉNÉRALE DES
FINANCES PUBLIQUES

Table des matières

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 a 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 a 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. Un état de l'art des méthodes d'analyse de logs existante a été compilé, ainsi qu'une analyse des fichiers de logs disponibles. Ensuite, une implémentation des algorithmes a été faite à l'aide d'une réécriture de l'architecture du projet.

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. A state of the art of existing log analysis methods was compiled, as well as an analysis of available log files. Then, an implementation of the algorithms was made using a rewritten project architecture.

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

Les mots succédés d'une astérisque au cours du document
sont définis dans les glossaires en fin du mémoire.

Introduction

Le projet `analyse-logs` a été développé par deux précédents apprentis de la Direction Générale des Finances Publiques (DGFiP)*, Rémi Grison puis Léa Lebert. Cependant, le projet ayant été développé il y a déjà 3 ans, une question s'est posée sur l'évolution du domaine. De nouvelles recherche, et de nouveaux algorithmes ont été publiés entre temps, et pourraient faire évoluer le projet. C'est ici que nait le sujet de ce PFE. Ce mémoire détaille le processus d'évolution de ce projet dans l'objectif de le tenir à jour sur l'état de l'art.

J'ai travaillé sur cette problématique en binôme avec Robin Gries, un stagiaire en dernière année de master à l'université de Nantes. L'organisation du sujet entre nous ainsi que les bénéfices et difficultés à travailler en équipe seront expliquées dans ce document.

Ce sujet de PFE s'intègre dans la stratégie d'innovation du Service des systèmes d'Information (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 du SI*. Dans ce cadre, une organisation proche de celle d'un projet de recherche a été suivie. Pour cela, nous avons dans un premier temps 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*. Dans un second temps, nous avons développé une nouvelle structure logicielle, pour ensuite implémenter, tester et comparer les différents algorithmes retenus.

Chapitre 1

La DGFiP, administration en évolution constante

La DGFiP* est une administration française née de la fusion de la Direction 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. La DGFiP compte aujourd'hui près de 95.000 employés titulaires, la faisant une des plus grandes administrations françaises.

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 SI*. Malgré une interaction indirecte avec le domaine métier des dinances 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 jusque aujourd'hui. L'informatique est donc au centre de cette administration, autant pour les agents en interne que pour les utilisateurs externes. On compte par exemple plus de 280 millions de visites cumulées sur le site `impots.gouv.fr`, et 83% des impôts de particuliers payés de façon dématérialisée.

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.

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. Le bureau est encore spécialisé en plusieurs divisions, chacune en charge de projets spécifiques. La Division des Applications des Professionnels 1 (DAP1)*, chapeautée par Olivier Blanc (mon tuteur), est à la fois une aide technique sur le domaine du logiciel, ainsi qu'une division détachée des projets principaux, permettant de développer des projets transverse*. Un des plus gros projets de cette division est le «cadriciel», une base logicielle Java qui supporte de nombreuses applications du bureau.

Depuis le début 2019, il existe aussi à la DGFiP une cellule innovation, elle-même sous la tutelle d'Olivier. Son objectif est de mettre en avant des technologies innovantes et des projets expérimentaux, pour permettre l'évolution des logiciels faces aux normes de développement. Les alternants de l'IMT embauchés par Olivier sont rattachés à cette cellule, leur permettant de mener des projets modernes et formateurs. De plus, les projets d'innovation et de modernisation s'inscrivent bien dans une démarche d'ingénieur, nous permettant de guider leur déroulement dans son entièreté.

Chapitre 2

Notions techniques

Ce PFE est un projet très scientifique, et va donc beaucoup dans la technique. En effet, l'objectif étant de trouver de nouveaux algorithmes d'analyse de logs* et de les mettre en place, une compréhension en détail du fonctionnement de ces techniques était nécessaire. Les notions dans ce chapitre peuvent donc être utiles à la compréhension des points les plus techniques de ce rapport. La lecture de la section 2.1 au minimum est conseillée, car elle explique en détail la structure des logs, qui sont centraux à ce sujet de PFE.

2.1 Que sont les logs ?

Le terme «log» est tiré de l'anglais et signifie à l'origine «journal», tel un journal de voyage. Il fait référence à 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 boîtes noires dans l'aviation.

Les logs informatiques découlent de cette définition, décrivant le chemin d'exécution d'une application ou même d'un système d'exploitation. Ces fichiers de logs sont séparés ligne par ligne en entrées de logs, souvent appelées «un log». Un log contient des informations communes à chaque entrée, ainsi que le message de log qui est la partie variable. Les informations communes sont en général 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 débugger 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 à ces informations, un développeur (ou quelqu'un chargé de la maintenance) sera capable de retrouver des événements qui ont eu lieu ainsi que de remonter les causes d'anomalies. C'est un processus 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.

2.2 Réseaux de neurones

Un réseau de neurones est un modèle de traitement de données, inspiré du fonctionnement réel des neurones présents dans notre cerveau. Il est au fondement de l'apprentissage profond*, dû à sa simplicité (relative) et sa versatilité. De nombreuses évolutions ont été produites sur ce concept, mais le principe reste toujours le même.

Un réseau de neurones est utilisé pour apprendre une fonction mathématique, avec un nombre donné d'entrées et de sorties, a partir d'exemples de couples {données, résultats}. Différentes architectures sont propices a apprendre différents types de fonction, avec par exemple les réseaux en convolution qui peuvent classifier des images, ou bien les réseaux antagonistes génératifs (GANs en anglais) qui sont capables de générer des images. Je vais ici décrire un réseau de neurones dense, qui est sa forme la plus basique. Il est théoriquement capable d'effectuer de nombreuses tâches, mais est plus adapté à des tâches simples.

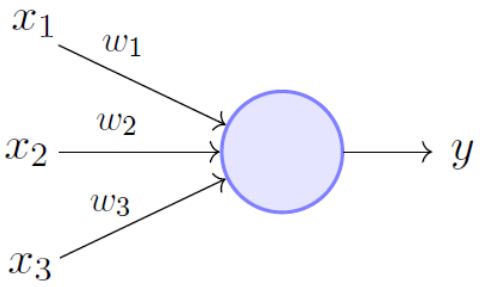

FIGURE 2.1 – Structure d'un neurone

Le neurone (ou perceptron) est l'atome d'un réseau de neurone. Il consiste en 3 éléments (voir figure 2.1), qui sont :

- Les entrées (sur le schéma x_1, x_2 et x_3) représentent les valeurs que le neurone reçoit et à partir desquelles il peut «agir».
- Les poids (w_1, w_2 et w_3) sont des facteurs de multiplication pour leurs entrées respectives.

- L'activation (y) ou sortie, est calculée comme la somme des entrées pondérées par leur poids, à laquelle on applique une fonction d'activation ϕ, soit : $y = \phi(x_1w_1 + x_2w_2 + x_3w_3)$.

Une fonction d'activation permet de transformer la valeur de poids pour la contraindre. La plus simple possible est la fonction en «marche» qui vaut $\phi(x) = 0$ si $x < 0$, et $\phi(x) = 1$ sinon. Les plus communes sont la fonction ReLU (pour Rectified Linear Unit) $\phi(x) = max(0, x)$ et sigmoïde $\phi(x) = \frac{1}{1+e^{-x}}$.

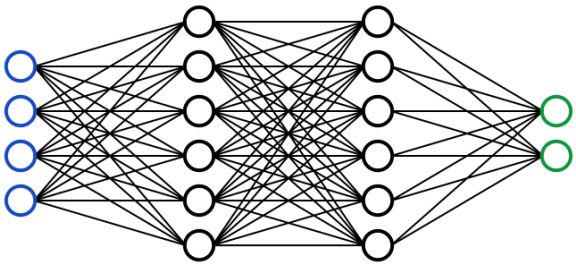

FIGURE 2.2 – Structure d'un réseau de neurones

Pour faire un réseau de neurones, comme son nom l'indique, il suffit de connecter plusieurs de ces neurones. Plus spécifiquement, on va créer des couches de neurones, avec pour chaque couche un nombre de neurones défini. On relie ensuite les activations de chaque neurone d'une couche aux entrées de chaque neurone de la couche suivante (voir figure 2.2). La première couche est la couche d'entrée, dans laquelle les données sont entrées dans le réseau. Ensuite, les neurones s'activent couche par couche (appelées couches cachées), en utilisant les valeurs d'activation de la couche précédente, jusqu'à la dernière couche de neurones. Cette dernière est la couche de sortie, et on utilise les valeurs d'activation de ses neurones comme résultat. Ce processus est appelé «propagation avant».

A partir de jeux de données d'entrainement, il est possible d'utiliser les données pour effectuer cette propagation avant dans le réseau. Seulement, par défaut, le réseau est initialisé avec des poids aléatoires, et le résultat sera donc très éloigné du résultat attendu. On peut alors utiliser le résultat attendu pour calculer l'erreur du réseau, soit la différence entre le résultat attendu et le résultat du réseau. Ensuite, on peut estimer l'importance de l'erreur provoquée par chaque neurone de l'avant dernière couche, et ainsi de suite couche par couche, cette fois en arrière. On appelle ce processus «propagation arrière» ou «rétropropagation». La méthode utilisée est la descente du gradient, qui est trop complexe pour être détaillée dans ce document, mais permet de diminuer l'erreur globale du réseau et donc d'améliorer ses prédictions.

En appliquant cet algorithme de nombreuses fois à l'aide de résultats connus d'un grand jeu de données, le réseau de neurones va pouvoir être ensuite utilisé comme outil de calcul ou de

classification en utilisant les motifs appris durant son entraînement. Le jeu de données utilisé est donc très important pour la qualité et la performance du réseau. Il nous faut à la fois de la quantité mais aussi de la variété, pour prendre en compte tous les cas de figure que le réseau pourra rencontrer durant son utilisation.

2.3 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 utile à eux seuls, ils sont très utiles pour les comparer entre eux. Voici des exemples communs de ce qui est possible, en notant $v(mot)$ le vecteur sémantique d'un mot :

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

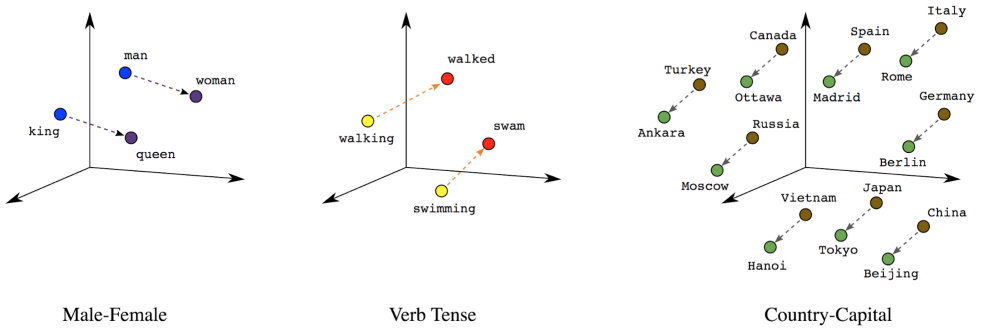

FIGURE 2.3 – Exemple de relations entre vecteurs sémantiques

Ces vecteurs sont générés par un processus relativement simple basée sur de l'apprentissage automatique*. Je vais brièvement expliquer ici deux méthodes, présentées en plus de détail dans le papier word2vec [9]. La figure 2.4 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.

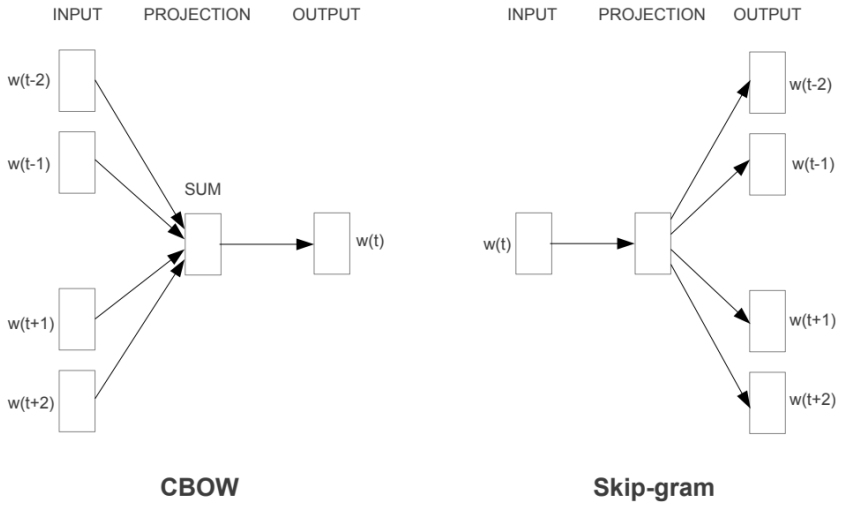

FIGURE 2.4 – Architectures de vectorisation de word2vec

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-entrainer un modèle, mais c'est très rare.

2.4 Forêt d'isolation

La forêt d'isolation, décrite en 2009 [8], 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 a 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 2.5.

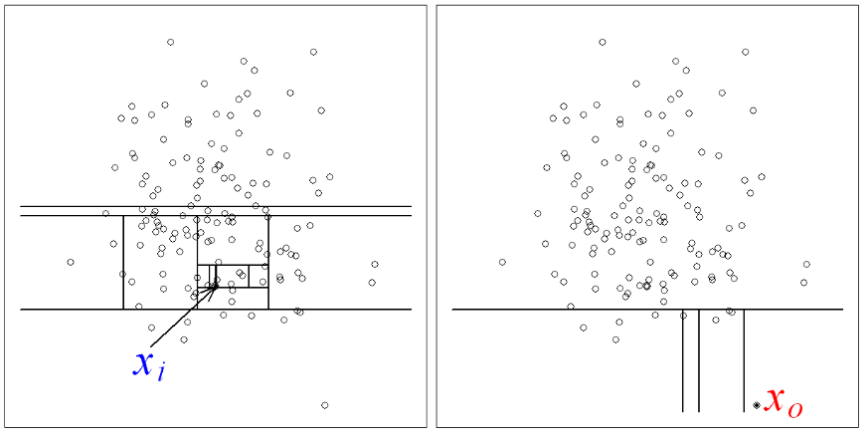

FIGURE 2.5 – Fonctionnement des forêts d'isolation

Chapitre 3

L'existant : le projet analyse-logs

3.1 Contexte du projet

L'informatique possède un rôle central au sein de la DGFiP*, rempli par le SI* et ses nombreuses applications métier. 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 aux 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 avéré 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 est parti du principe que les logs sont le langage parlé des applications. Si un modèle était capable d'apprendre ce langage, il serait possible de détecter des fautes, qui représenteraient des anomalies de l'application.

Rémi Grison, apprenti à la DGFiP en 2019, partant de cette idée, a trouvé le papier DeepLog [4], 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 son sujet de PFE. Léa Lebert à 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. [4]. Je vais ici détailler le fonctionnement de DeepLog ainsi que l'implémentation qui a été faite par Rémi.

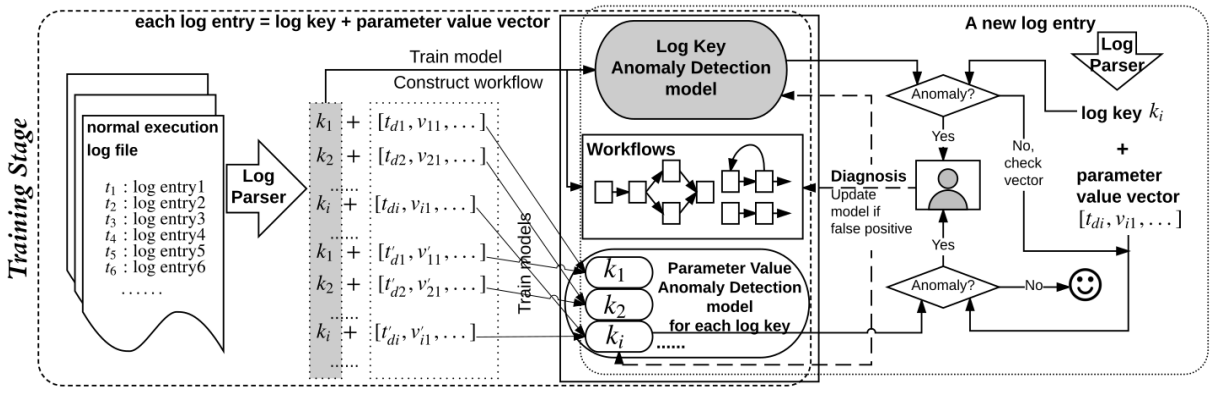

FIGURE 3.1 – Structure de DeepLog

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 [3], 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 [7]. Cet algorithme a 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.

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.

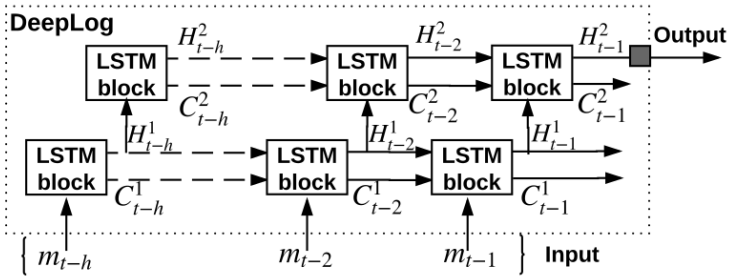

FIGURE 3.2 – Structure d'un réseau LSTM empilé

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

Les ids des logs* sont utilisés : On entraine 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 enchainement 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 logs 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 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 lesquels j'avais aidé Léa au moment de son PFE.

Blockly

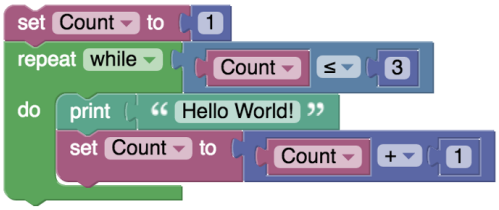

FIGURE 3.3 – Exemple de programme avec Blockly

Blockly est une bibliothèque 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 programmation Scratch, son éditeur visuel est basé sur Blockly.

Olivier et Léa ont fait le choix d'utiliser cette bibliothèque 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. En plus de permettre la substitution des algorithmes par d'autres alternatives, il permet aussi de simplifier l'utilisation du projet pour les futurs utilisateurs. En effet, il est possible avec Blockly de facilement et intuitivement créer et tester des modèles, adaptés aux besoins des différentes applications de la DGFiP*. Un exemple de modèle construit avec l'interface blockly est présenté en figure 3.4.

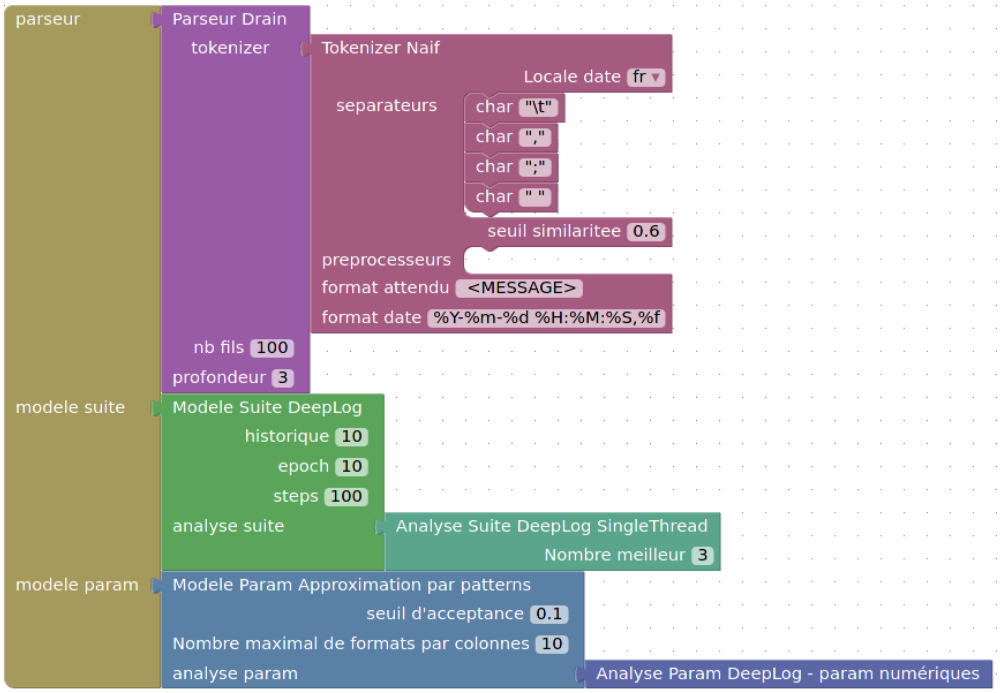

FIGURE 3.4 – Modèle d'analyse construit avec Blockly

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 alors envisageable de remplacer une étape par une autre similaire.

Le premier algorithme qui a été choisi est les «forêts d'isolation» [8], 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 2.5). 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. J'ai pu assister Léa sur la fin de son PFE, pour implémenter des nouveaux algorithmes au sein de l'architecture. J'ai aussi pu apporter une aide technique sur la partie apprentissage automatique* du projet après avoir suivi un MOOC 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 bibliothèque 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 bibliothèque telle que

scikit me permet tout d'abord une meilleure robustesse et performance des algorithmes, mais aussi une certaine généricité. En effet, les algorithmes du même type ont tous la même structure de fonctions et cela permet très facilement de 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

Gestion et évaluation du projet

4.1 Organisation du projet

La planification et les aléas

Comme je l'ai mentionné plus haut, Robin a démarré son stage avant mon retour d'école. Il a donc fallu lui planifier ses tâches à réaliser avant mon arrivée, afin d'utiliser efficacement tout le temps disponible pour le projet. Mon premier travail à donc été, avant la période d'école de Janvier, de faire une première planification du projet et tout particulièrement les tâches de Robin sur la période de février.

Ce premier planning effectué est disponible en annexe 3. Différents points sont intéressants à noter à son sujet. Tout d'abord, la possibilité pour moi d'un stage à l'étranger était toujours envisagée. Cependant, suite à une absence de réponse de l'entreprise que j'avais contacté, j'ai décidé de suivre l'exercice de remplacement à la place. Cela a permis de libérer du temps pour la réalisation du projet.

Les sprints définis dans ce planning étaient relativement libre dans leur contenu. L'idée principale était de développer, tester et comparer les algorithmes retenus pendant l'état de l'art tout en essayant de suivre un mode agile.

Notre méthode de travail avec Robin était très collaborative. Nous avons beaucoup discuté au fil du projet pour régler des problèmes, ou mieux comprendre des points particuliers. Cette méthode collaborative est très intéressante dans le cadre d'un projet de recherche, car sa complexité était élevée, et nous n'aurions pas pu avancer aussi bien sans ces échanges importants.

Le premier point qui est venu impacter le planning est ma sous-estimation de la charge de travail liée à la partie académique du projet. En effet, la recherche peut être un processus lent et fastidieux. Dans le cadre de la méta-analyse (analyse de nombreux papiers de recherche pour en tirer des conclusions), ce n'est que plus vrai, nécessitant d'abord de trouver les bon papiers du domaine, puis de les comparer et en tirer des résultats.

Cet allongement des tâches de Robin m'a permis de commencer le développement de la réécriture du projet, qui n'était pas du tout prévue dans le planning à l'origine. Cette réécriture m'a pris beaucoup de temps, et a impacté le planning.

A la moitié du projet, un point à été fait pour évaluer notre progression ainsi que l'évolution de la planification. Les changements de planning effectués sur la première moitié du projet, ainsi que les changements dans la planification future, sont reflétés dans le second planning en annexe 4. Les sprints ont été remplacés par des tâches précises, nous avons donc abandonné l'agilité. Je vais donc détailler les obstacles à l'utilisation des méthodes agiles, ainsi que les causes dans notre projet, dans la partie suivante.

Les freins à l'agilité

L'agilité est une approche à la gestion de projet née du manifeste agile, basée sur 4 valeurs simples : («>» représente «plutôt que»)

- Individus et interactions > Processus et outils
- Logiciel fonctionnel > Documentation exhaustive
- Collaboration avec le client > Négociation contractuelle
- Adaptation au changement > Exécution d'un plan

De ces points naissent différentes méthodes telles que Scrum, qui décrit un cadre plus précis sur la mise en place et la réalisation de cette gestion de projet. Des «sprints» sont mis en place, qui séparent le temps du projet en cycles courts itératifs (en moyenne autour de quelques semaines). A chaque sprint, différentes tâches sont attribuées aux développeurs et une démonstration de ce qui a été réalisé est faite à la fin de ces sprints.

Les méthodes agiles s'appliquent très bien au développement informatique et c'est majoritairement dans ce domaine qu'elles sont utilisées. Mais une étude publiée en 2013 [6] met en avant les différents facteurs qui peuvent freiner leur mise en place. Le papier nous montre le fait que ces freins à l'agilité sont en majorité causés par la culture et la structure trop traditionnelle des organisations concernées. Certains des points mentionnés dans ce papier s'appliquent en effet à notre situation, mais des nuances sont présentes dans le contexte d'un projet de recherche. Nous étions aussi un binôme, et nous n'avions pas de méthodologies précédentes pour nous faire former des aprioris.

Un des problèmes mentionné dans le papier est le temps de formation, qui à été un de nos plus gros freins. Si j'ai effectivement étudié les méthodes agiles en cours, je n'ai jamais réellement eu l'occasion de les mettre en pratique dans mes projets. L'utilisation assez limitée de l'agilité dans les projets de la DGFiP*, ainsi que mon manque d'implication dans d'autres projets, ont limité mon expérience de terrain avec ces méthodes, et il était donc difficile pour moi de les mettre en place.

Pour reprendre les différents principes agiles, il est intéressant de noter que le premier point a bien été mis en place sur le projet. En effet, la relation entre moi et Robin, ainsi qu'entre nous et Olivier, était un des points cruciaux au bon déroulement du projet. Des processus détaillés n'ont pas été mis en place pour ceci, mais aussi en partie car il est compliqué de formaliser un processus de recherche et d'innovation.

Un problème pour la mise en place de l'agilité à été le second point. En effet, la mise en place d'un logiciel fonctionnel n'est pas une évidence dans un cadre de recherche. Si le principe agile parle de documentation pour le logiciel, je pense que cette idée peut aussi prendre en compte la documentation en amont du logiciel. Ce processus de documentation a finalement été au cœur du projet, au dépens de la partie technique et implémentation. Cependant, il est important de noter que cette partie n'était pas négligeable. Elle a duré le temps nécessaire pour trouver, comprendre, sélectionner et compiler les différents algorithmes. Ce genre de choses ne sont pas vraiment prises en compte dans des méthodes comme Scrum, qui se concentrent plutôt sur des projets de développement pur.

Pour ce qui est de la collaboration avec le client, on peut considérer Olivier comme le client de ce projet. Ce qui a manqué pour mettre en place ce point, notamment comme dans Scrum avec le «sprint review», était une production régulière et présentable de fonctionnalités. Scrum se base sur le développement d'un MVP, auquel on ajoute des fonctionnalités complètes a chaque sprint, et le client est capable dans la review de les tester et de faire évoluer les besoins. Ce n'est pas que les négociations contractuelles ont pris le pas sur la communication, comme il n'y en a pas eu non plus ; le point s'est juste difficilement appliqué au projet.

Le dernier point résonne énormément avec le projet. En effet, comme détaillé dans la partie précédente, le planning était notre source d'organisation, et si des sprints étaient prévus par la suite, une partie des tâches était très fixée dans le temps. Au fur et à mesure du déroulement, il est devenu évident que le planning ne correspondrait pas à l'effort réel demandé pour certaines tâches. Nous avons donc fait évoluer le planning durant le projet pour qu'il reflète la différence avec le réel. C'est là que mettre en place une méthode agile aurait pu vraiment nous aider. Certes, nous n'aurions pas eu de façon d'approximer le temps total du projet ainsi que des différentes tâches, et c'est souvent un des problèmes qui se posent pour mettre en place ces méthodes. Mais cela nous aurait permis de ne pas essayer de suivre un planning, et plutôt de se concentrer sur le plus important : ce qu'il y a à faire (soit le backlog).

Finalement, je trouve dommage de ne pas avoir pu mettre en place l'agilité dans le projet, et j'aurais dû y faire plus attention, car un projet de recherche se porte en réalité très bien à son utilisation. En effet, le fait de partir d'une problématique relativement vague, et d'incrémentalement améliorer les objectifs ainsi que le projet est exactement le but de l'agilité. Il est intéressant de noter qu'il existe des façons d'appliquer les méthodes agiles avec un seul développeur [11] [10], et elles pourraient être étendues a une équipe de deux personnes aisément.

4.2 Évaluation économique du projet

Coût du projet

Le projet étant assez limité en terme de moyens mis en place, son coût le sera aussi, principalement réduit au coût du personnel affecté au projet. Le tableau 4.1 décrit une approximation du coût humain de ce PFE. Les congés payés et périodes de cours ne sont pas retirée de la durée totale, en considérant que ce temps est tout de même situé durant la réalisation du projet ainsi que rémunéré.

En comptant une rémunération d'alternant de plus de 21 ans en dernière année d'alternance, mon salaire correspond à 78% du SMIC, soit 1 283,55€ brut par mois. En prenant en compte les charges patronales à hauteur de 42% du salaire brut, le coût pour l'entreprise atteint 1 822,64€ par mois pour l'entreprise. Cela correspond à un taux pour l'entreprise de 84,12€ par jour ouvré.

Pour ce qui est d'un stagiaire comme Robin, le coût est moindre. La gratification pour un stage de plus de deux mois est de 3,90€ par heure, ce qui revient a 27,30€ par jour ouvré. Ce montant est exonéré de charges sociales et patronales, il n'y a donc pas de surplus a considérer dans ce calcul.

	Début	**Fin**	**Durée**	**Coût**
Robin Gries	Février	Juin	105 jours ouvrés	2 866,5€
Oscar Gloaguen	Mars	Août	127 jours ouvrés	10 683,24€
Olivier Blanc	Février	Août	26 jours ouvrés ($\sim$1 jour/semaine)	?
Total				13 549,74€

TABLE 4.1 – Coûts humains de ce PFE

Il reste à prendre en compte les coûts matériels du projet. Mais il s'avère que ces coûts reviennent finalement à 0. En effet, aucun matériel n'a été affecté spécifiquement pour ce projet : les PC que moi et Robin utilisaient nous ont certes été affectés, mais ce sont des PC qui seront réutilisés suite à notre départ. De la même façon, nous utilisons des systèmes d'exploitation libres et gratuits (Fedora pour ma part), ainsi que des logiciels gratuits ou sous licence étudiante (par exemple la suite JetBrains). Le coût logiciel est donc lui aussi 0. Nous n'utilisons aussi pas de serveurs, tels que pour des machines virtuelles, étant donné la teneur encore expérimentale du projet.

Ce projet est donc relativement peu coûteux pour la DGFiP, en partie par le fait qu'il est principalement composé d'un stagiaire et d'un alternant, mais surtout de l'absence de coûts matériels supplémentaires. Son intérêt dépendra alors énormément du retour sur investissement qu'il amène, détaillé dans la partie suivante.

Retour sur investissement

Chapitre 5

Processus scientifique et réalisation 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. Robin 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 le déroulement du projet et les différents points important que nous avons rencontrés.

5.1 État de l'art de l'analyse de logs

À l'arrivée de Robin sur le projet, j'étais encore en période école. Il a donc commencé le travail d'étude de l'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 d'accessibilité des chercheurs, 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 [2] publié par Huawei et l'université de Hong Kong en 2021. Ce document très récent, étant publié environ 6 mois avant le début du projet, 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 intéressant. Nous avons pu comme cela mieux comprendre les différents papiers trouvés, faire des choix sur l'implémenta-

tion et des hypothèses à propos des résultats.

Cette recherche nous a mené à choisir les algorithmes que nous souhaitions 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

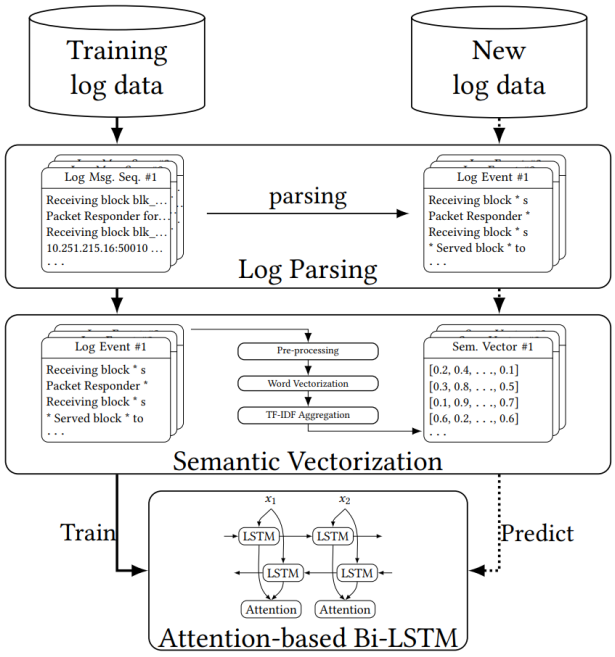

FIGURE 5.1 – Structure de LogRobust

LogRobust [12] 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 les problèmes des anciennes méthodes de l'état de l'art en proposant une solution robuste. L'architecture du modèle est détaillée sur la figure 5.1.

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 2.3 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-entrainement du modèle.

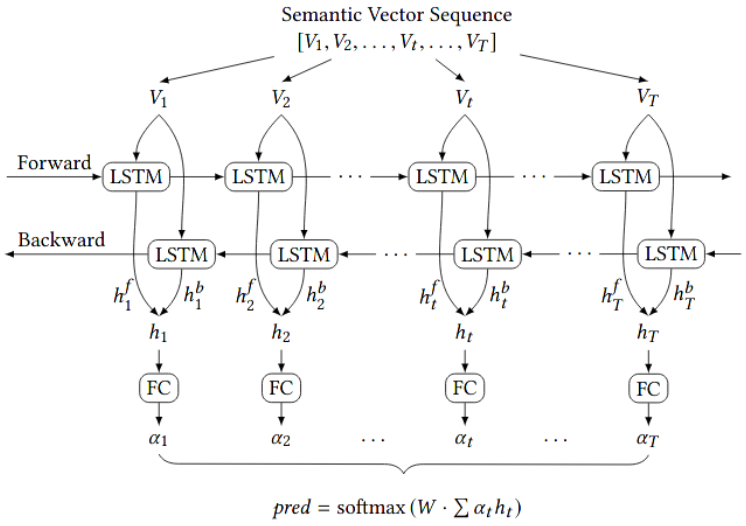

FIGURE 5.2 – Structure d'un LSTM bidirectionnel

Une autre différence notable comparé à DeepLog est l'utilisation d'un bi-LSTM* avec un mécanisme d'attention (voir figure 5.2). Le fonctionnement d'un bi-LSTM est similaire au LSTM utilisé 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.

Le mécanisme d'attention simule le concept d'attention humain. Par exemple, quand on lit une phrase, on est capable de savoir quels 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 quels 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%. L'attention est mise en valeur car elle permet de garder des 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-entrainer 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 [5] est appelée «AutoEncoder» dans le reste du document, mais elle utilise en réalité 2 réseaux de type auto-encoder ainsi que des forêts d'isolation.

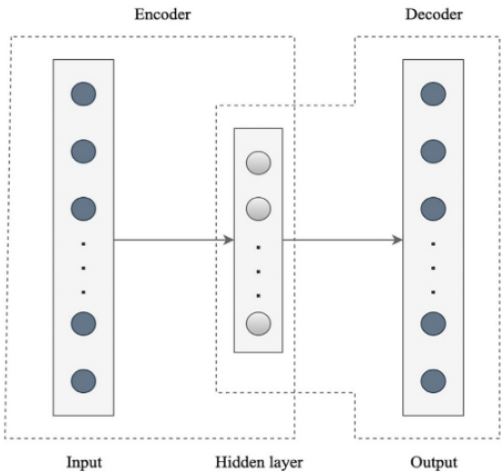

FIGURE 5.3 – Structure d'un réseau autoencoder

Le plus important pour comprendre ce modèle est le fonctionnement d'un auto-encoder. C'est encore une fois un réseau de neurone, avec une structure simple en «double entonnoir» (voir figure 5.3). 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.

La structure complète du modèle est détaillée dans la figure 5.4. Le premier auto-encoder est entrainé 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

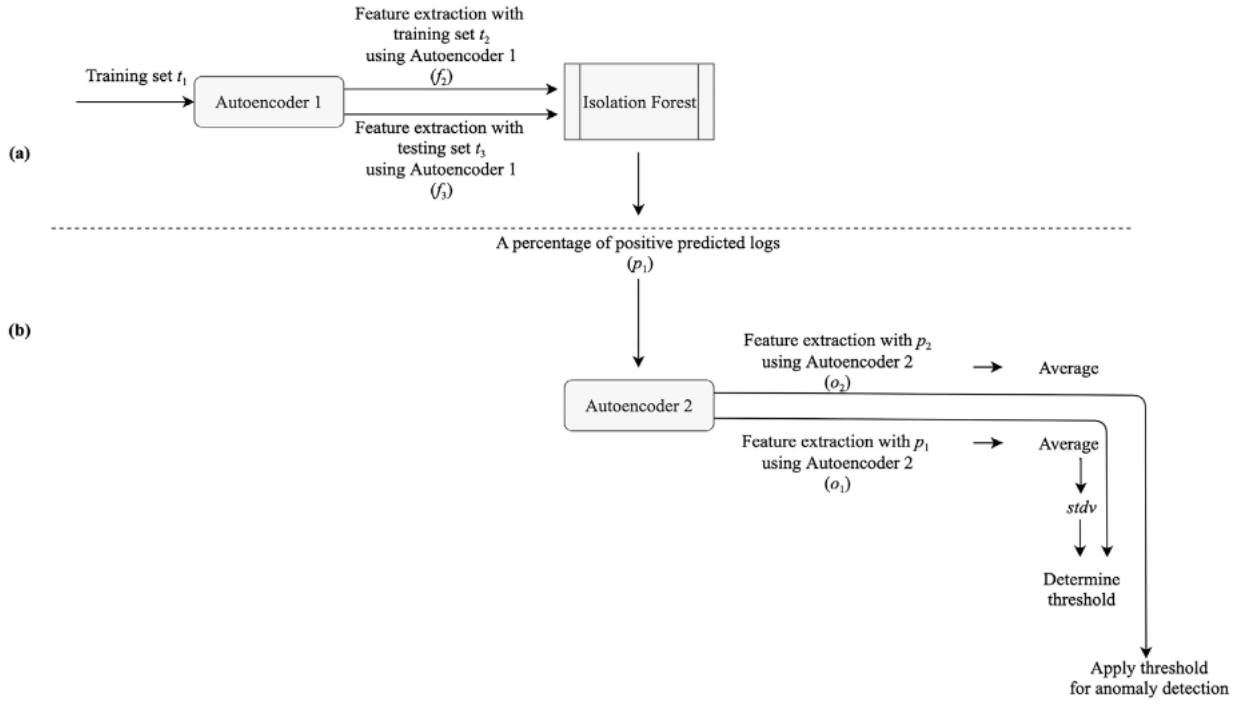

FIGURE 5.4 – Structure du modèle «AutoEncoder»

le processus et améliore la détection. Enfin, une seconde passe de détection d'anomalies est faite avec un second auto-encoder, 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 [2] 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.

5.2 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 logs 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 duquel 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*. 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. Un notebook permet de séparer du code Python en différents blocs exécutables séquentiellement et séparément pour du traitement de données et des visualisations.

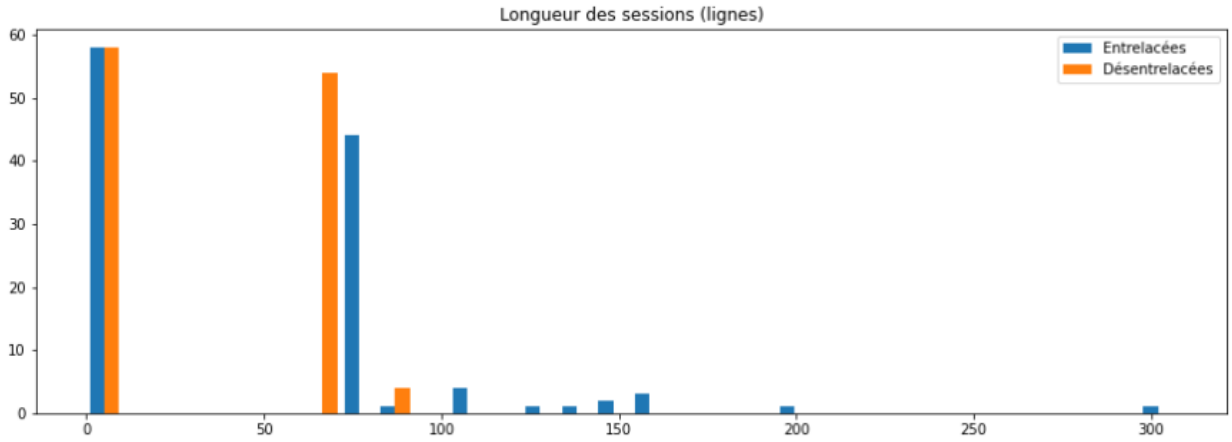

FIGURE 5.5 – Longueur des sessions dans les logs MEDOC

La première donnée sortie de cette analyse est la longueur des sessions, entrelacées ou non pour permettre une comparaison (figure 5.5). 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. L'entrelacement des sessions est assez minime en moyenne, mais cause 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 logs. Enfin, on observe aussi l'existence d'une session qui comporte plus de 1000 lignes même en étant désentrelacée.

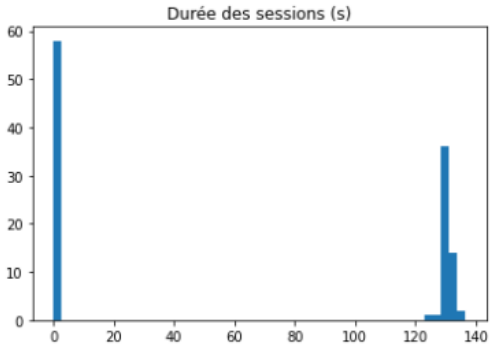

FIGURE 5.6 – Durée des sessions

Une analyse de la durée des sessions a aussi été effectuée (voir figure 5.6). 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 évènements. Catalina est un composant de Tomcat, le moteur de serveur utilisé à la DGFiP. Catalina est en fait l'implémentation du moteur des «servlets» Tomcat (voir article wikipédia Servlet pour plus d'informations). Ces logs* représentent des fermetures de sessions Catalina, qui ne sont 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 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 5.7.

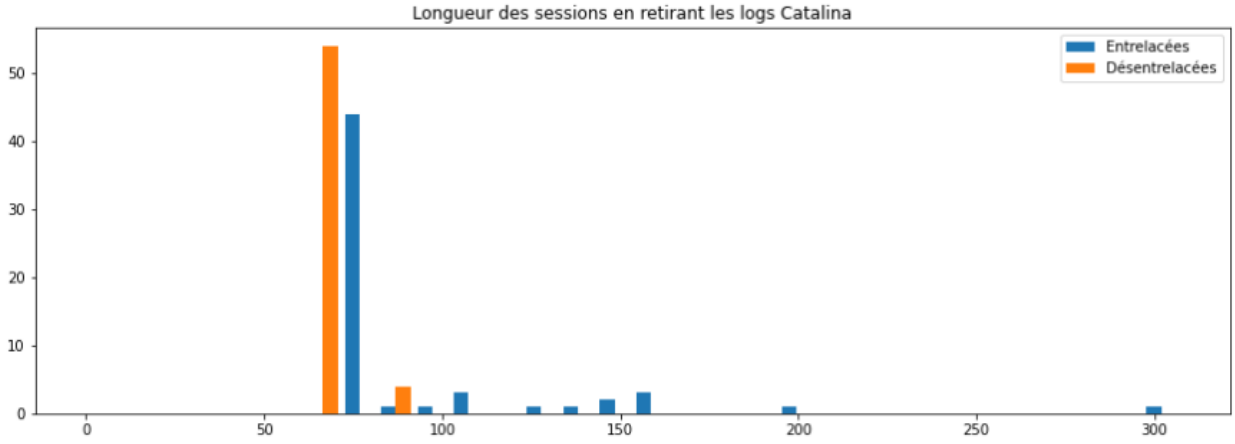

FIGURE 5.7 – Longueur des sessions en retirant les logs Catalina

La 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 5.8 détaille les pourcentages de logs appartenant a une session, ainsi qu'une session Catalina.

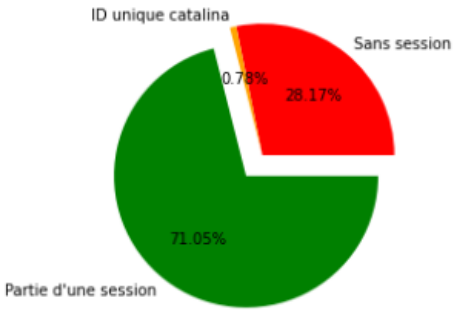

FIGURE 5.8 – Proportions de logs appartenant à des sessions

5.3 Développement du nouveau projet

Durant la partie scientifique du projet, et ayant déjà travaillé sur le projet de Rémi et Léa, différents points d'évolutions sont devenus apparents. Le projet est très large et il ressemble à un projet auquel on aurait rajouté des modules au fur et à mesure, sans planification au départ, ni de temps passé à repenser cette structure. Le problème qui se pose ici est celui de la dette technique, qui s'accumule sur les projets à grande échelle si elle n'est pas prise en compte.

J'ai alors fait le choix de faire une réécriture du projet et surtout de repenser entièrement son architecture. L'objectif était de la rendre flexible aux changements et surtout ajouts, pour permettre au projet de grandir et évoluer avec les différents projets qui l'utiliseraient dans le futur. Principalement, on doit être capable d'aisément ajouter des algorithmes différents proposant différentes analyses et caractéristiques, pour le futur du projet, mais aussi et surtout pour pouvoir les implémenter et tester plus facilement.

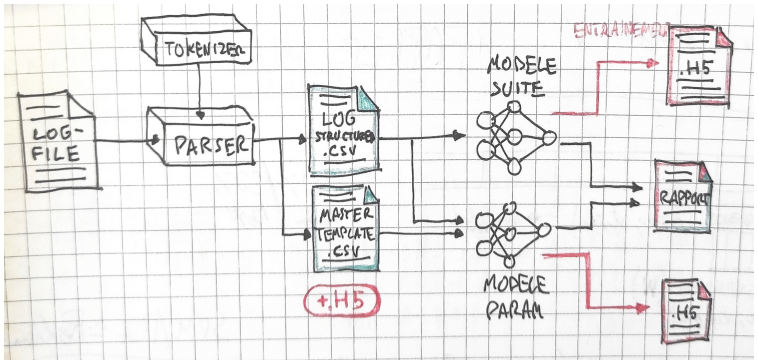

FIGURE 5.9 – Structure des modèles du projet original

La figure 5.9 décrit la structure des modèles d'analyse des logs* dans le projet original. Elle est très similaire à celle de DeepLog (voir figure 3.1), étant donné que le projet a été à l'origine simplement une implémentation de cet algorithme. Un problème se pose quand on souhaite implé-

menter des algorithmes différents de DeepLog dans leur architecture. C'est d'ailleurs pour cette raison que Léa n'a pas implémenté d'autres algorithmes, mais simplement certains des blocs fonctionnels dans cette structure.

L'idée était donc de réécrire le projet, en partant non pas d'un algorithme en particulier mais d'une architecture permettant d'implémenter les différents algorithmes envisagés, et plus si nécessaire. Celle-ci devrait bien sûr inclure DeepLog, donc la structure parseur → analyse suite / analyse paramètres doit être comprise dans l'architecture. En utilisant des structures de classes et d'héritage, beaucoup d'abstractions peuvent être faites.

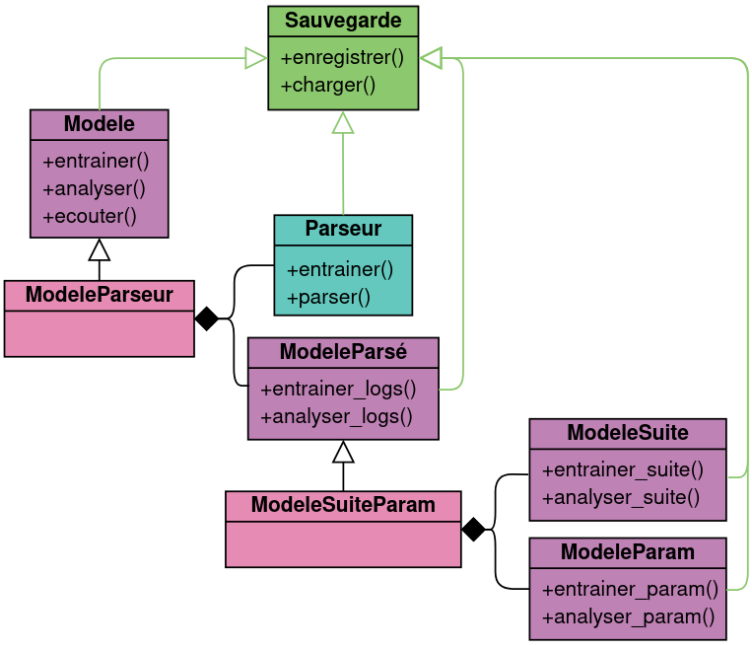

FIGURE 5.10 – Nouvelle architecture pour les modèles d'analyse

La figure 5.10 détaille la structure de classe pour les modèles de la réécriture. On peut voir beaucoup de similarités avec la structure précédente, mais celle-ci est déjà plus formalisée, et surtout il est envisageable d'y rajouter des composants. Cela est dû à la séparation des types de modèles, ainsi que leur abstraction. Un modèle complet est capable de s'entraîner, d'analyser un fichier de logs*, et d'écouter des logs en direct. On n'a pas besoin de connaître son fonctionnement interne si on l'utilise juste à haut niveau. Ce principe est utilisé à tous les niveaux de l'architecture, par exemple pour la sauvegarde : tous les types de modèles, ainsi que les parseurs, doivent pouvoir être enregistrés et chargés, et c'est donc défini par l'interface «Sauvegarde».

Cette architecture permet ensuite l'implémentation des différents algorithmes retenus. La (ré)-implémentation de DeepLog est un point à prendre en compte, pour permettre son utilisation au sein de la nouvelle version ainsi que sa comparaison avec les autres algorithmes. Une première

version à été produite à l'aide de la bibliothèque open-source Python `deeplog`. Elle se base sur une implémentation assez rudimentaire du modèle à l'aide de PyTorch, une bibliothèque d'apprentissage automatique* assez reconnue. Un module «`ModèleDeeplogParsé`» à donc été développé, en passant les différentes méthodes directement à la bibliothèque. Ensuite, en intégrant une bibliothèque implémentant Drain dans un module `Parseur`, il est possible de construire un `ModèleParseur`. Ce dernier est alors utilisable comme décrit plus haut.

L'implémentation faite par Rémi avec Tensorflow a ensuite été en partie réutilisée, de façon à être intégrée dans le modèle de classes. Ici, on implémente des modules de type `ModèleSuite` et `ModèleParam`, de la même façon que dans le projet original. Ces modèles sont regroupés dans un `ModèleSuiteParam`, qui est combiné avec le module Drain pour obtenir, encore une fois, un modèle complet et utilisable. L'implémentation des autres algorithmes se fait de la même façon, en intégrant les algorithmes dans les différents modules qui correspondent pour former un modèle complet.

Chapitre 6

L'humain contre la machine

L'intelligence artificielle est une technologie qui a connu une grande expansion durant cette décennie, et comme toute technologie, a connu des éloges ainsi que des critiques. Mais plus elle est présente dans notre vie, plus les peurs liées à son utilisation se font ressentir. En grande partie, les gens ont peur d'une chose : la perte de leur travail aux dépens d'une machine. Nous allons étudier cette thèse avec l'aide du premier chapitre du livre «Le futur du travail» [1] par J.S. Carbonell, chercheur en sciences sociales du travail.

Dans ce chapitre, Carbonell met en avant le fait que le travail n'est pas simplifié (et ne disparaît pas) par l'automatisation, bien au contraire. Les machines ne nous remplacent pas autant qu'elles changent notre relation au travail, mis en évidence par le fait que malgré ce mouvement d'automatisation, il n'y a pas de rupture du marché du travail pour cette raison. En effet, derrière l'automatisation se cachent de nombreux métiers que Carbonell appelle «invisibles», le revers de la médaille étant la précarité de ces travailleurs de l'ombre.

En fonction des industries, les effets et les mécanismes de l'automatisation sont vastes et variés, et le remplacement du travail n'en est qu'une unique facette. Dans la suite de cette partie, nous allons détailler les conséquences que l'automatisation peut apporter au travail, telles que décrites par Carbonell.

La première, mais surtout la plus «fantasmée», est le remplacement du travail humain par le travail d'une machine. L'exemple ici est donné de l'atelier de peinture d'une chaîne de production automobile : «Un robot peut enregistrer toutes les 20 millisecondes la position d'un ouvrier peintre, ce robot est ensuite capable de reproduire exactement la séquence de l'opération qui à été enregistrée et, ainsi, de confisquer des séquences entières du travail ouvrier.» Le remplacement est d'autant plus facile que le travail est répétitif, l'exemple des lignes de production d'usine est donc tout particulièrement pertinent. On pourra mentionner «Les temps modernes» de Charlie Chaplin, qui critique de manière humoristique le remplacement des ouvriers dans ces usines de production.

Carbonell explique la cause de l'attention portée a cette conséquence à l'industrie sidérurgique, pour laquelle un levier de réduction de la masse salariale était l'automatisation de substitution. Elle apportait un très grand gain de temps et de moyens aux industries, avec des suppressions de jusqu'à deux tiers des effectifs.

La seconde conséquence est décrite comme un processus de «déqualification/requalification» du travail. La suppression de tâche ne pousse pas forcément à la suppression de postes, et il est possible de redistribuer le travail restant parmi les employés. Selon les observations de Carbonell, cela provoque un mouvement symétrique où «une partie de la main d'œuvre est déqualifiée ou déclassée, une autre est requalifiée ou reclassée.»

La déqualification est facile à expliquer : La division du travail crée des tâches faciles à automatiser, et ce qui reste pour les employés est du travail le plus simple et sans intérêt possible. L'économiste Benjamin Coriat, cité par Carbonell, parle de «taylorisation assistée par ordinateur».

La requalification est provoquée par la nature même de l'automatisation, à travers la maintenance, la programmation, ou même la surveillance des machines. Cela peut avoir des effets positifs, par exemple des ouvriers de production qui deviennent des ouvriers d'entretien, mais aussi des effets négatifs, tels que le licenciement des salariés moins qualifiés. En effet, le coût de formation pour la requalification d'un employé n'est pas négligeable, et il peut être parfois plus intéressant d'embaucher de nouveaux travailleurs déjà qualifiés. Il est aussi possible que de requalifier certains puisse déqualifier d'autres, revenant a l'effet symétrique mentionné plus haut.

La troisième conséquence de l'automatisation est appelée «intensification» du travail. Carbonell décrit par ce mot différents facteurs : «l'augmentation des rythmes de travail, la dégradation de la santé et de la sécurité au travail, la hausse de la charge physique [et mentale]». Il explique ce point par les motivations des employeurs à l'automatisation : si on automatise les taches simples et répétitives, les salariés se retrouvent avec les tâches qui apportent plus de valeur pour l'entreprise, et qui demandent donc plus de travail. On optimise non pas seulement les tâches qui sont automatisées, mais aussi celles qui ne le sont pas, sur lesquelles les employés vont pouvoir se concentrer.

La quatrième et dernière conséquence est le contrôle, ou l'augmentation du pouvoir de la direction sur les salariés. Il est possible, à travers les machines et l'informatique, de surveiller les temps de travail, les périodes de pauses, la productivité des employés... De plus, certaines de ces technologies sont mises en place spécifiquement dans le but de contrôler ou surveiller le lieu de travail.

Il est important de noter que ces points sont des conséquences de l'automatisation par les directions d'entreprise, mais pas nécessairement des objectifs de cette dernière. Si les révolutions technologiques précédentes (machine à vapeur, électricité, internet) amenaient ce même genre

de questionnement on voit aujourd'hui qu'elles ont principalement provoqué la disparition de certains métiers pour en créer d'autres. Selon l'INSEE, le secteur de l'énergie représente aujourd'hui près de 180.000 emplois, soit presque 1% de l'emploi en France, qui n'existait pas avant cette révolution.

Cette nuance peut aussi s'appliquer à ce qu'on appelle la «4e révolution industrielle», soit les nouvelles technologies de l'informatique telles que l'intelligence artificielle. Cette «révolution» est cependant différente des précédentes, car si ces dernières semblaient mettre en péril le travail manuel d'usine, celle-ci touche aussi le travail intellectuel de bureau. Seulement, il faut garder en tête qu'à l'instar des précédentes, le vrai élément mis en jeu sera notre relation au travail et la nature des métiers exercés. En effet, l'intelligence artificielle est très dépendante du travail de nombreux programmeurs, que ce soit les développeurs du modèle qui définissent son architecture ou leurs utilisateurs qui l'entraînent pour une tâche spécifique. L'intelligence artificielle n'est rien sans données, et ces données sont produites, compilées et traitées par l'humain.

Il est alors possible de mettre en relation cette étude avec notre projet de PFE. Il inscrit dans cette «4e révolution industrielle», et les craintes associées peuvent le suivre vis a vis de l'automatisation du travail.

Ici, son but premier est effectivement la substitution à une tâche auparavant tout particulièrement humaine. La détection d'erreur dans des logs* peu formalisés est une tâche qui demande de la réflexion et de l'expérience, autant dans le domaine en général et avec l'application en particulier. Cette substitution n'est pas une substitution totale comme nous avons pu en parler plus haut, mais bien une substitution d'une tâche particulière et répétitive. On peut alors espérer, en libérant les développeurs de cette tâche, les requalifier, au moins une partie de leur temps, leur permettant d'effectuer des tâches plus stimulantes. L'intensification du travail des développeurs est une conséquence possible de la mise en place de ce projet, mais il faut prendre en compte la difficulté et la lourdeur de la tâche de détection d'erreur.

Conclusion

Ce mémoire décrit différentes facettes d'un sujet de PFE très complexe, qui comporte à la fois des concepts techniques et scientifiques poussés, ainsi que des problématiques organisationnelles et humaines modernes. Je pense pouvoir dire que ce projet a été un succès, dressant un état de l'art complet et comparatif des différents algorithmes d'analyse de logs* et en proposant une implémentation flexible et accessible. Malgré les difficultés rencontrées au court du projet, nous avons su coopérer et le mener jusqu'aujourd'hui.

Ce projet est voué à être repris et évoluer dans le futur, soit par de futurs alternants de IMT Atlantique, soit par des développeurs intéressés par le sujet au sein de la DGFiP*. J'espère que ces futurs évolutions permettront l'adoption de ce projet au sein des équipes de développement. Cette initiative mettrait en avant la valeur de l'innovation et des technologies modernes pour les projets de la DGFiP.

Bibliographie

[1] Juan Sebastián Carbonell. Le futur du travail. 2022.

[2] Zhuangbin Chen, Jinyang Liu, Wenwei Gu, Yuxin Su, and Michael R. Lyu. Experience report : Deep learning-based system log analysis for anomaly detection, 2021.

[3] 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.

[4] 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.

[5] Amir Farzad and T. Aaron Gulliver. Unsupervised log message anomaly detection. *ICT Express*, 6(3) :229–237, 2020.

[6] Taghi Javdani Gandomani, Hazura Zulzalil, Abdul Azim Abdul Ghani, Abu Bakar Md Sultan, and Mina Ziaei Nafchi. Obstacles in moving to agile software development methods ; at a glance. *Journal of Computer Science*, 9(5) :620, 2013.

[7] 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.

[8] Fei Tony Liu, Kai Ting, and Zhi-Hua Zhou. Isolation forest. In *Eighth IEEE International Conference on Data Mining*, pages 413 – 422, 01 2009.

[9] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[10] Sibonile Moyo and Ernest Mnkandla. A novel lightweight solo software development methodology with optimum security practices. *IEEE Access*, 8 :33735–33747, 2020.

[11] Anna Nyström. Agile solo-defining and evaluating an agile software development process for a single software developer. 2011.

[12] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng 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.

DAP1 Division des Applications des Professionnels 1: Première division logicielle du BSI-3*, avec une mission de support et d'innovation, dirigée par Olivier Blanc..

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.

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 a 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 a développer des algorithmes généraux basés sur l'apprentissage de données, s'opposant a 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

Annexes

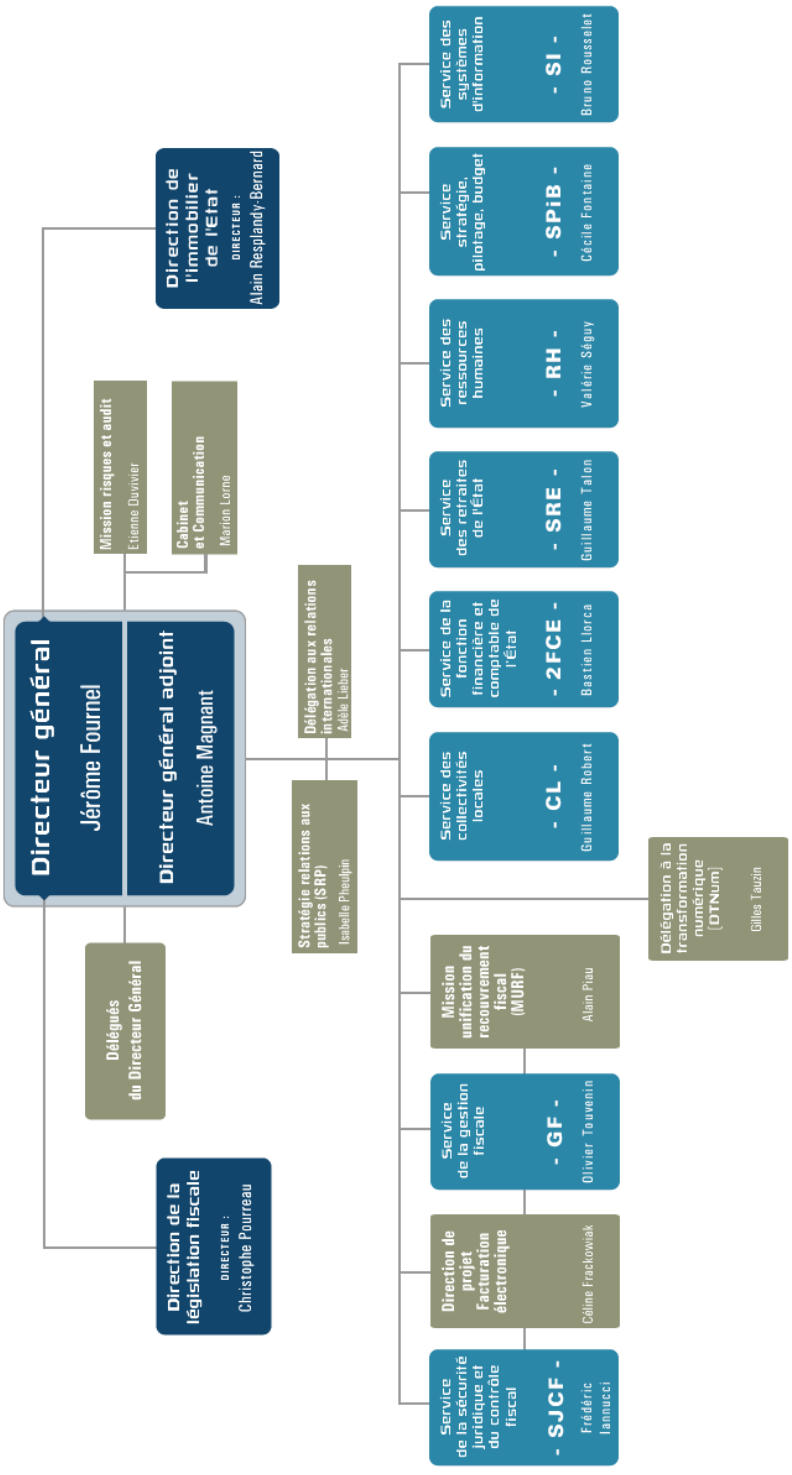

FIGURE 1 – Organigramme des services de la DGFiP

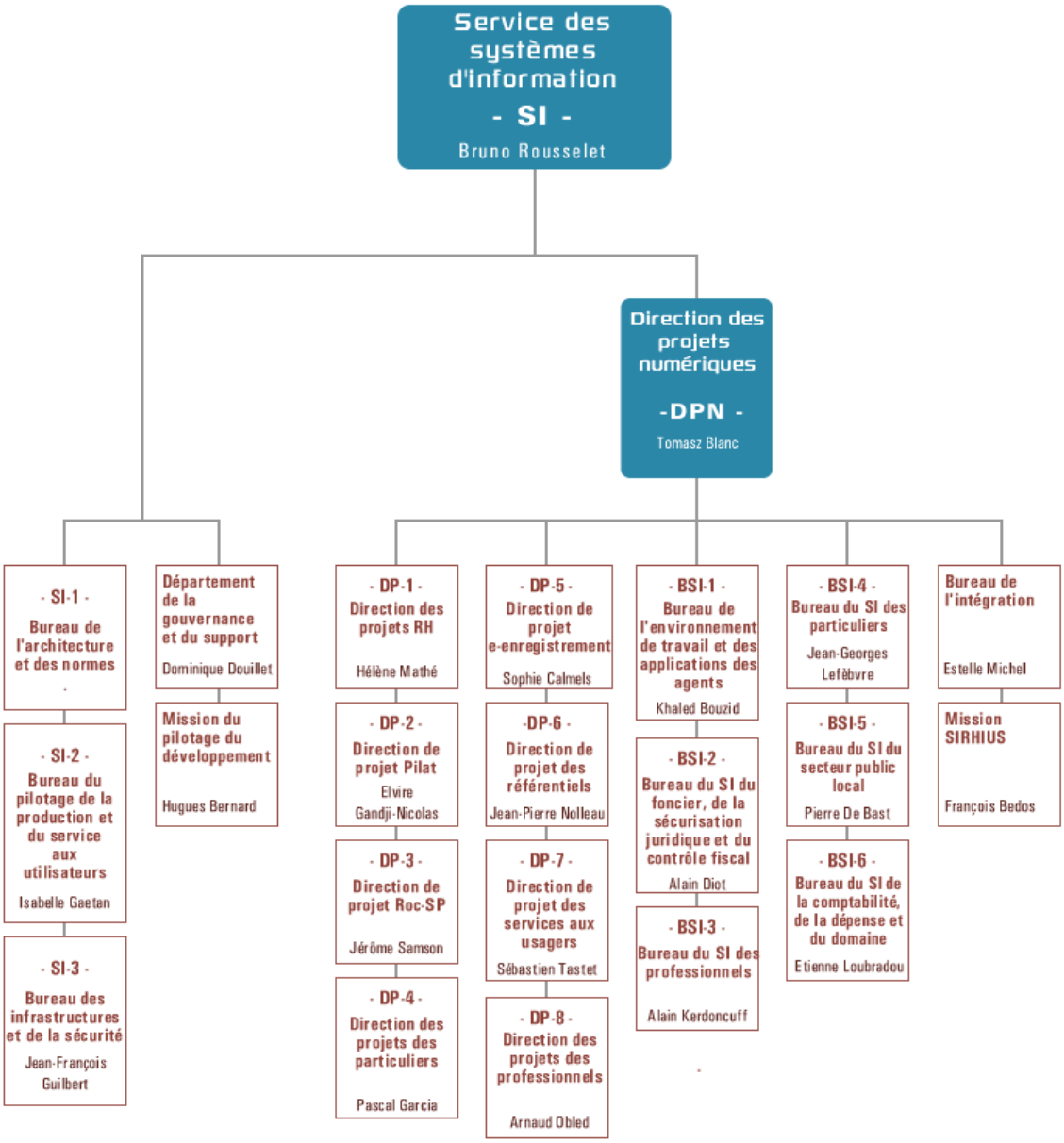

FIGURE 2 – Organigramme des bureaux du SI

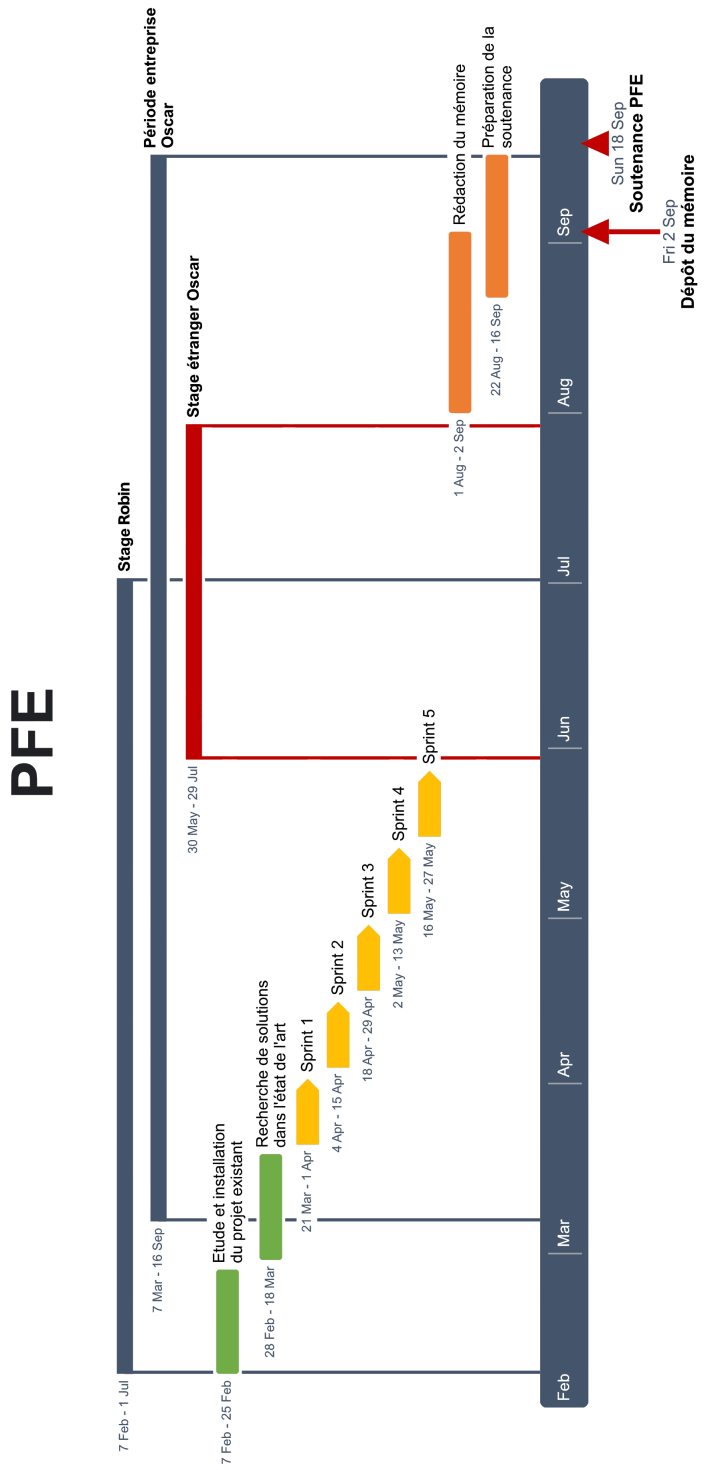

FIGURE 3 – Planning original du projet

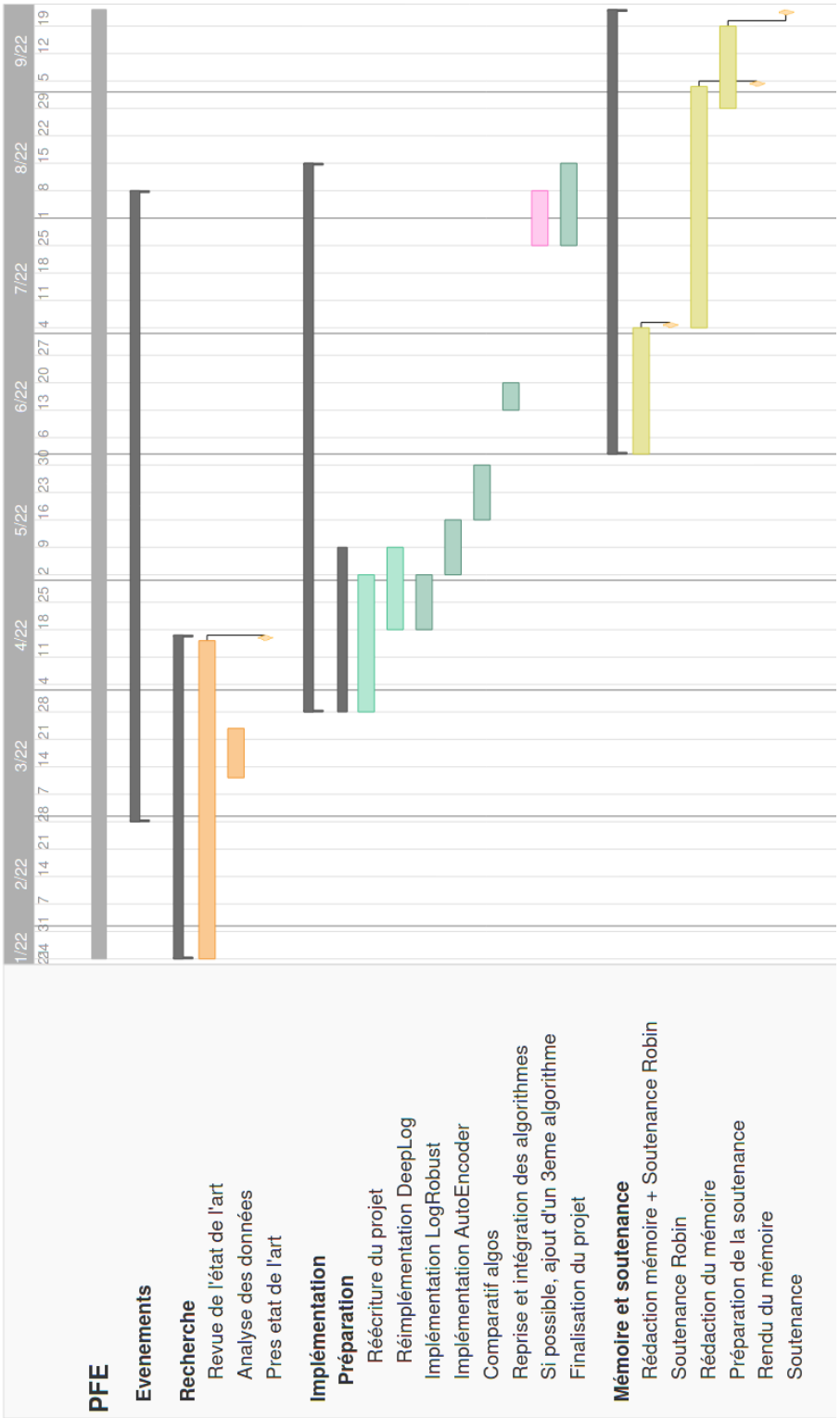

FIGURE 4 – Planning mis à jour au milieu du projet