{<L3/>}
INFORMATIQUE

Rapport de stage

Bourdeau Tom – 22205482

Tuteur : Delalandre Mathieu

Du 23/04 au 27/06

université de TOURS
Faculté des sciences & techniques

Remerciements :

Je tiens à remercier M.Delalandre pour son encadrement tout au long de mon stage, pour son aide dans le développement des différentes applications et pour sa pédagogie lors des réponses à mes nombreuses questions.

J'aimerai également remercier le LIFAT pour son accueil au sein de ses locaux, et à l'entièreté de son équipe pour leurs retours lors des différentes réunions de présentations, et même lors de nos échanges au quotidien.

Déclaration plagiat :

Je soussigné(e), Tom BOURDEAU (22205482), certifie être l'auteur de ce rapport de stage, avoir moi-même effectué ces recherches, et atteste que ce rapport de stage n'a pas déjà été remis dans le cadre d'un autre diplôme. Tout énoncé emprunté au travail d'un/d'une autre (avec ou sans changements mineurs) et cité dans ce mémoire apparait entre guillemets, et cet emprunt fait l'objet d'une référence complète et précise. Je suis conscient que le plagiat peut se traduire par l'invalidation de ce rapport de stage mais aussi, dans les cas les plus graves, par une exclusion de l'Université. J'affirme également qu'à l'exception des emprunts dument reconnus, ce mémoire constitue mon propre travail.

Sommaire

Balises d'utilisation des outils d'IA générative :

Titre et sigle du cours :	Rapport de stage de Licence Informatique – L3 SLS
Session, date :	07/2025
Nom de la personne enseignante :	T.BROUARD / N.LABROCHE

Niveau d'utilisation autorisé par la personne enseignante

NIVEAU 0	NIVEAU 1	NIVEAU 2	NIVEAU 3	NIVEAU 4
	☒	☒	☐	☐
Utilisation interdite	Utilisation limitée	Utilisation guidée	Utilisation balisée	Utilisation libre

Pour le domaine :

☒ Disciplinaire

☒ Langues

Nom de la personne étudiante : Bourdeau Tom

Titre de la production : Rapport de stage

D'après :

D'après :

	Utilisations	O	N	IAg utilisées	Requêtes utilisées[1]
NIVEAU 1	**Domaine disciplinaire** : Informatique				
	S'inspirer	☐	☒		
	Générer des idées	☐	☒		
	Explorer un sujet pour mieux le comprendre	☒	☐		Aide pour veille technologique
	Générer du matériel pour son étude	☒	☐		Aide pour veille technologique
	Autre utilisation : Cliquez ou appuyez ici pour entrer du texte.	☐	☒		
	Domaine des langues :				
	Identifier ses erreurs et se les faire expliquer	☐	☒		
	Reformuler un texte	☐	☒		
	Générer un plan pour aider à structurer un texte	☐	☒		
	Traduire un texte	☐	☒		
	Autre utilisation : Cliquez ou appuyez ici pour entrer du texte.	☐	☒		
NIVEAU 2	Analyser des contenus	☐	☒		
	Obtenir une rétroaction	☐	☒		
	Évaluer la qualité de son travail à partir de critères	☐	☒		
	Demander à être confronté relativement à ses idées, à sa démarche	☐	☒		
	Diriger les processus de résolution de problèmes	☐	☒		

	Utilisations	O	N	IAg utilisées	Requêtes utilisées[1]
	Autre utilisation : Cliquez ou appuyez ici pour entrer du texte.	☐	☒		
NIVEAU 3	Résumer ou rédiger des parties d'un texte	☐	☒		
	Générer un texte ou un modèle d'une production et l'adapter	☐	☒		
	Réaliser des calculs mathématiques	☐	☒		
	Produire du code informatique	☒	☐		Production des codes complexes et refactoring
	Résoudre des problèmes complexes	☐	☒		
	Répondre à une question	☒	☐		Utilisé pour toutes sortes de questions
	Générer des images, ou autres contenus multimédias	☒	☐		Génération schémas présentation orale
	Autre utilisation : Cliquez ou appuyez ici pour entrer du texte.	☐	☐		
NIVEAU 4	Cliquez ou appuyez ici pour entrer du texte.	☐	☐		
	Cliquez ou appuyez ici pour entrer du texte.	☐	☐		
	Cliquez ou appuyez ici pour entrer du texte.	☐	☐		

I. Introduction

1. Présentation du LIFAT

Le LIFAT (Laboratoire d'Informatique Fondamentale et Appliquée de Tours), dirigé par Hubert CARDOT, est le laboratoire de recherche en informatique de Tours. Il est chargé des recherches dans divers sujets tels que : la conception ou le développement de modèles, de méthodes et d'algorithmes; la créations de ressources et de logiciels d'extraction des informations de différentes sources; l'exploitation de ces données extraites afin d'en tirer des connaissances (via l'intégration d'interaction homme machine); et la résolution de problèmes d'optimisation combinatoire afin d'obtenir un compromis optimisé entre la qualité des résultats et le temps de calcul.

Il est actuellement composé de 88 membres, parmi lequel des enseignants-chercheurs, des doctorants et des post-doctorants, répartis en trois équipes de recherche :

- Bases de données et Traitement du langage naturel (**BdTln**)

- Recherche Opérationnelle, Ordonnancement et Transport (**ROOT**)

- Reconnaissance des Formes et Analyse d'Images (**RFAI**)

Ces équipes distinctes sont reliées par leurs domaines d'applications :

- La santé et le handicap : Que ce soit avec le CHU de Tours ou avec les équipes de l'INSERM (Institut National de la Santé et de la Recherche Médicale), le LIFAT a participé a la création d'outils d'aide technique pour personnes en situation de handicap physique ou atteints d'autisme, ainsi que pour l'optimisation des flux, l'analyse d'images pour l'aide au diagnostic, et la fouille visuelle de données médicales, etc...

- Les données massives et le calcul haute performance : Le LIFAT a pour vocation de résoudre des problématiques autour des infrastructures de stockage et d'accès aux données, de calcul GRID/CLOUD, d'extraction, d'analyse/structuration de données, d'exploitation, de visualisation.

- Les humanités numérique : Avec ses nombreux partenariats présents et passés réalisés en lien avec le CESR (Centre d'Etudes Supérieures de la Renaissance) et le laboratoire CITERES (Cités, TERritoires, Environnement et Sociétés), le LIFAT est également chargé de la création de structures des bases de données, de la numérisation 3D et de la reconnaissance de formes.

2. Présentation du RFAI

Le RFAI (Reconnaissance des Formes et Analyse d'Images) est donc l'équipe en charge des recherches sur les méthodes permettant de construire et d'exploiter des représentations de

haut niveau sémantique à partir de diverses données d'entrée. Ses domaines d'expertises sont donc multiples : le machine learning, le data mining et l'image processing. Ces recherches se placent à plusieurs niveaux dans l'analyse de processus : bas niveau (filtering, segmentation, detection of interest points…) mais aussi haut niveau (matching, classification, indexing…). Le RFAI travail en particulier sur :

- Les méthodes interactives : intégration de saisie de l'utilisateur et de connaissances préalables dans le processus de reconnaissance, data mining visuel dans des environnements de réalité virtuelle embarquée.

- Les Graph-based methods : Construction de représentation pour le Matching et la classification.

- Machine Learning : Méthodes incrémentales, algorithmes biology-inspired, Réseau de neurones convolutif, apprentissage par ensemble.

- Image processing : filtrage, segmentation, méthodes variationnelles, optimisation discrètes et continues.

Ces différentes spécialités permettent l'analyse de tous types de données : Les documents numérisés, les images et vidéo d'un environnement réel, des nuages de point 3D, des images médicales et de séries temporelles. Cette équipe possède un vaste réseau de partenariat, que ce soit avec des entreprises Françaises et à l'Internationale, mais aussi avec d'autres laboratoires à travers le monde. (Voir Remercîments)

Parmi ces laboratoires on retrouve l'Université HCMUT (Ho Chi Minh City University of Technology), université vietnamienne avec laquelle le LIFAT collabore sur le projet auquel j'ai été assigné.

3. Présentation du projet Station TV

La Station TV est une station de calcul parallèle appliquée à la télévision développée depuis 2017 par le laboratoire de recherche du LIFAT et de l'école Polytech Tours. Son but est de capturer et de traiter automatiquement des flux télévisuels de la TNT grâce à deux machines (sur lesquels nous reviendrons en détail) : la DELL PowerEdge T640 spécialisée dans le traitement en temps-réel de flux vidéo et la DELL 5820 en charge de la capture vidéo de flux TNT, eux aussi en temps-réel. L'association de ces deux machines a pour but de permettre le déploiement d'applications en lien avec les flux télévisuels, d'analyse d'image et vidéos et de traitement (direct ou différé) de ces sources.

Entre 2017 et 2025 de nombreux contributeurs, étudiants et chercheurs, se sont succédé pour améliorer le fonctionnement de la station, développer des outils utilisant la capture vidéo et pour perfectionner/stabiliser ces derniers. Parmi ces applications nous pouvons citer des outils permettant : la détection de publicités, le fact checking lors notamment de débat politiques, l'annotation automatique du contenu, la capture smart de vidéos et d'images, l'analytics TV et le scraping de données TV.

Lors de ce stage, le projet auquel j'ai été rattaché est l'amélioration de la méthode NER (Named Entity Recognition) de données audio TV. Plus particulièrement dans le traitement logiciel et physique de ces données. En effet, à terme le projet a pour vocation à retranscrire en direct et à l'écrit des contenus télévisuels audios, pour de multiples usages tels que le filtrage par thèmes ou le fact checking. Pour se faire, les retranscriptions se doivent d'être aussi intègres que possible et la vitesse de calcul doit être à minima aussi rapide qu'à celle ou les données sont produites. Ainsi, l'ensemble des taches qui m'ont été confiées ont ainsi été dans ce sens : Observer, évaluer, tester et déployer des solutions permettant une retranscription aussi rapide et propre que possible.

II. Capture audio large échelle et transcription

1. Problématique de transcription audio

L'objectif principal du projet qui m'a été confié est de permettre la mise en place d'un outil de retranscription écrite de paroles entendues dans des programmes télévisuels. A terme l'outil sera déployé sur des flux en temps réels, c'est à dire que l'outil aura la capacité de retranscrire en direct ce qui est dit dans les programmes de la TNT.

L'utilisation de cet outil permettrai l'implémentation d'autres outils, notamment en facilitant l'accès aux contenus des émissions, pour par exemple réaliser du fact-check live, ou même en facilitant la recherche par mots clé, afin de trouver plus rapidement toutes les émissions et programmes qui auraient mentionné un sujet en particulier. Aussi, cela pourrait permettre de créer une alternative au stockage volumineux des fichiers audio, réduisant considérablement l'espace nécessaire à la sauvegarde de certains contenus.

Pour réaliser cet outil, nous nous devons de respecter une contrainte fondamentale qui deviendra rapidement la raison de toutes ces recherches : La vitesse de retranscription des fichiers se doit d'être aussi rapide que possible. En effet, comme nous l'expliqueront dans la suite, la taille des données sur lesquels nous travaillons sont colossales : On parle de flux de plusieurs To par semaines. Ainsi même 5% de vitesse gagnée par une solution nous fait économiser des temps de traitement non négligeable, aussi bien en temps de traitement qu'en coûts énergétiques et en temps de disponibilité des machines (puisque plusieurs groupes utilisent les machines de capture).

Une grande partie de ce projet consistera donc à chercher et développer une technologie optimale qui garantirai une vitesse suffisante, sans pour autant compromettre la qualité de la retranscription, qui doit rester exploitable.

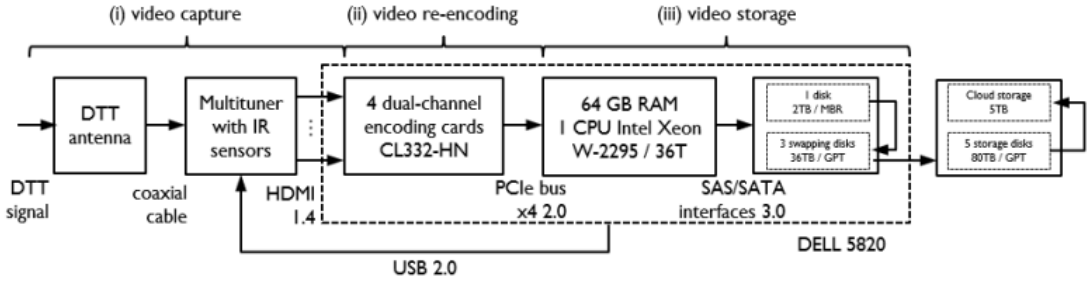
2. Présentation des machines à disposition

Afin de mettre en place une telle technologie, nous nous devons de posséder des machines à la fois de capture mais aussi de traitement de flux TNT. Pour se faire, le RFAI a à disposition deux machines :

- La DELL 5820, qui se charge de la capture vidéo : Cette machine est capable de capturer le flux vidéo et audio pour 8 chaines en simultané, avec une qualité HD et en 30 images par secondes. Elle est reliée à une antenne TV Hertzienne (située sur le toit de Polytech) et possède une configuration spécialement prévue pour la capture audio et vidéo de flux de la TNT : elle comprend un RackStation de 8 Tuners TNT, reliées à 4 cartes d'acquisition et d'enregistrement avec deux entrées chacune et de 8 émetteurs infrarouges Phidget. A cela s'ajoute 64Gb de ram (via deux barrettes de 32Gb) ainsi qu'un CPU INTEL Xeon W-2295 de 36 cœurs. La capture a la particularité d'être asynchrone, c'est à dire qu'on capture l'audio et la vidéo dans deux fichiers distincts. En effet, lors de la capture synchrone on a pu se rendre compte que bien que le CPU soit très puissant, la machine ne possédant pas de GPU et le CPU ne proposant pas de support

HD-graphic qui aurait pu compenser, les CPU sont surexploités et cela génère l'apparition d'artéfacts. La solution à ce problème est donc la capture asynchrone, avec d'un côté la carte Avermedia CL332-HN qui capture la vidéo, et le CPU qui s'occupe de l'encodage audio via un encodage x264.

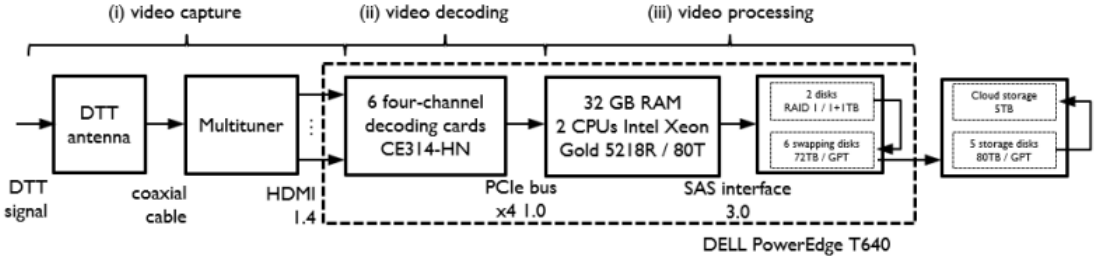
(Schéma composants DELL 5820)



Cette machine est spécialisée pour son utilisation première : Capturer et encoder du flux TNT. Ainsi, elle seule ne suffirait pas à traiter les flux, en parallèle de la capture, d'où la seconde machine.

- DELL PowerEdge T640 se charge elle du traitement en temps réel de flux vidéo. Elle possède également des composant lui permettant la capture de flux, mais dans ce projet elle ne sera utilisée que pour sa puissance de calcul. En effet elle possède 32Go de mémoire vive, un biprocesseur composé de deux CPU Intel Xeon Gold 5218R d'une puissance, pour chaque processeur de 20*2.1GHz. Cette puissance de calcul nous sera particulièrement utile lors des calculs, d'autant plus qu'elle permet l'exécution de 80 threads en simultané.

(Schéma composants DELL PowerEdge T640)



- Mon ordinateur personnel : Composé d'un Intel i5-1135G7 en 8 cœurs, de 8Gb de RAM, et à la différence des autres machines il est équipé d'un SSD de 258Go.

3. Présentation de la base de données

Afin de mettre en place ces outils, ainsi que les autres qui sont développés en parallèle, nous avions besoin des données dans un format réaliste et avec des données réelles. Ainsi "hello world" a été créée dans le but de tester nos données avec un échantillon très réduit, afin de tester sur des durées réalistes nos solutions. Nous avons donc capturé une semaine typique de flux TNT, afin d'imiter le contenu qu'à terme notre programme se devra de retranscrire. Voici donc les caractéristiques de la base : Tout d'abord, les chaines choisies pour cette base de test

sont au nombre de 7 (W9, Arte, C8, France 5, France 3, France 2 et TF1). La capture a duré une semaine, à raison de 12h par jours (de 10h à 22h). Afin d'éviter une latence trop grande causée par de la capture d'un flux trop long, chaque capture a été coupée en 3 (3*4h), toujours en capture asynchrone. La durée totale de la capture est ainsi de 588 heures. Ces fichiers sont répartis en 21 fichiers (x2 puisque asynchrone) par jours, pour un total de 147 (x2 puisque asynchrone) ; pour une qualité d'enregistrement en 720x576 (SD) en 30 images par secondes. Le total des fichiers fait 811 GB, mais puisque nous cherchons à retranscrire l'audio, seulement 62.9 GB nous intéressent pour la retranscription. Idéalement nous espérons trouver un moyen de traiter ces 588h en 12h maximum, afin d'avoir un flux de capture équivalent au flux de traitement de ces données.

4. Solutions de Speech-to-Text

Ainsi comme expliqué dans la problématique, l'objectif de l'outil est de retranscrire des fichiers audios en texte. Les solutions à ce problème n'étant pas ce qu'il manque, une veille technologique a dû être réalisée afin de trouver une solution optimale. Nos trois objectifs principaux sont donc :

1. **Analyser des fichiers audios et retranscrire ce qui est dit en texte.**
2. **L'outil doit permettre l'analyse de fichiers audios en Français à minima.**
3. **L'outil doit à terme pouvoir être déployable sur des flux en direct.**

Les sous objectifs, pas moins importants, nous permettrons de choisir l'outil que nous mettrons en place en fonction des détails. On peut par exemple citer la facilité d'installation et de déploiement, la volonté d'une solution open-source, une maintenabilité sur le temps via des mises à jour récentes, la gratuité de la solution et du fait que nous possédions deux machines surpuissantes (DELL PowerEdge T640 et DELL 5820) nous n'avons à priori pas besoin d'une solution en cloud.

Nos critères d'évaluations, eux sont simples. Premièrement, l'analyse qualitative : Est-ce que notre outil retranscrit bien ce qui est dit ? A-t-il des hallucinations ou des erreurs grotesques qui rendrait le résultat inexploitable ? Ensuite, l'analyse de débit : Est-ce que la vitesse d'analyse est suffisante par rapport à la vitesse à laquelle est produite la donnée à retranscrire ? Et enfin, notre PowerEdge étant configurée en Windows Server, l'outil sera-t-il mettable en place sur une telle configuration ?

Comme dis précédemment, les solutions ne manquent pas. Que ce soient des outils privés, open-source ou communautaires, nous pouvons trouver un grand nombre d'outils afin de retranscrire des audios en texte. Chacun possède sa spécialité, son cas d'usage et ses spécificités. La veille technologique a ainsi été longue, mais nous a permis de trier les différentes solutions et de trouver la plus optimisée pour notre usage :

Premièrement, les outils les plus performants en termes de temps de calcul sont les solutions cloud mises en place par des entreprises privées. Google Cloud Speech-to-Tex, Amazon Transcribe, Microsoft Azure AI Speech... Ces solutions sont extrêmement performantes mais à un certain coût, or possédant déjà une machine capable de réaliser des calculs complexes et

cherchant à éviter les outils payants, ces derniers ne correspondent pas aux critères que nous recherchons.

Nous avons donc cherché du côté des outils open-source disponibles, et ici aussi le choix est large. Quatre outils ont retenu notre attention :

- Mozilla DeepSpeech : Fort d'une API simple d'utilisation, et conçu pour un usage en temps réel, cet outil ne nous convient que moyennement à cause de l'absence de modèles prés entrainés en Français, et du manque de mises à jour depuis 2011.

- Kaldi : L'un des plus performant modèle de reconnaissance vocale, moins simple d'accès mais plus flexible dans son utilisation, n'est qu'utilisable en C++ limitant la facilité de déploiement.

- Vosk : Cet outil possède des modèles prés entrainés sur plus de 20 langues et un déploiement facilité par les multiples langages dans lesquels il est utilisable (Python, Java, C#, Node.js, C++, Go, Rust) n'est pas le choix optimal car il est spécialisé pour les appareils à ressources limitées (ex : systèmes embarqués) par sa faible empreinte mémoire et une utilisation efficace du CPU.

4. Whisper : Outil développé par OpenAI, il est le modèle le plus utilisé et de très loin. Ce modèle est entrainé sur un nombre colossal de données (plus de 680 000 heures) en multilingue, ce qui en fait l'outil le plus robuste même avec des bruits ou des musiques de fond. Il est utilisable via une API python et propose différents modèles plus ou moins performant (et donc couteux en CPU et/ou GPU).

Notre choix s'est donc porté sur Whisper.

III. Tests des performances de Whisper

1. Introduction et paramètres

Whisper est une api créée par OpenAI utilisable en ligne de commande et en python. Nous utiliserons la version python afin de pouvoir de faire passer plusieurs tests à cet outil. En effet, avant de mettre en place l'outil, nous souhaitons tester dans des conditions expérimentales plus légères nos outils et les différents paramètres que met en place Whisper.

Le fonctionnement de Whisper est simple, nous reviendrons dans les détails plus tard :

- 1. Chargement d'un modèle
- 2. Transcription du fichier
- 3. Ecriture du résultat de la transcription dans un txt

Pour évaluer nos applications de Whisper, nous utiliserons des métriques spécifiques que nous devons détailler avant d'entrer dans les détails :

- Temps de réponse : Durée totale durant laquelle s'exécute un processus.

- Débit (ou Throughput) : Correspond au $\sum_{i=1}^{n} \frac{D_i}{RT_i}$ avec n= nombre de fichiers, Di = durée du fichier i, RT = Response Time du fichier i. En multipliant le débit par la durée d'un fichier, on obtient la durée de retranscription d'un fichier.

- Taux CPU : Pourcentage d'utilisation du CPU.

- Consommation mémoire : Pourcentage d'utilisation de la RAM.

La base « hello world » mentionnée précédemment étant de taille conséquente, nous avons été dans l'obligation de créer plusieurs bases synthétiques permettant de réaliser les tests à moindre coût et de gagner du temps :

- **baseMod** : Il s'agit d'une base de données composée de 5 fichiers audio.mp3, respectivement de 4, 8, 12, 16 et 20 minutes d'un enregistrement provenant d'un journal TV. Le fichier de 8 est le résultat de la concaténation du 4min deux fois, celui de 12 3 fois etc.... La taille du fichier de 4min est de 3,66Mo, celui de 8= 2*3,66 etc.... Tous sont encodés en 124kbps.
- **baseMtr** : composé de 32 fois le fichier de 4minutes précédemment évoqué.
- **baseBatch :** composé de 32 dossiers, et en fonction des besoin chaque dossier est garni de 2, 3, 5, 14, 24 ou 32 fichiers de chacun 60min de durée (résultat de la concaténation du fichier de 4min 15fois)

2. Comparaison des modèles

Comme mentionné plus tôt, Whisper doit charger un modèle lors de l'étape 1. Cela s'explique par le fait que Whisper n'est pas un seul et unique modèle de retranscription. En effet, il existe 5 modèles que l'on peut mettre en paramètre de la fonction load_model().

```
model = whisper.load_model("small")
```

Les 5 modèles sont dans l'ordre : **tiny, base, small, medium et large**. Ces modèles sont rangés du moins au plus précis, avec bien sur la charge de travail CPU qui augmente en fonction de la précision de la retranscription.

Ainsi le premier test à mettre en place devra nous permettre de se faire une idée de quel modèle nous pourront mettre en place dans notre solution, à la fois en vérifiant le temps d'exécution mais aussi en comparant le résultat obtenu. Ce test a été réalisé sur ma machine personnelle.

Pour ce qui est de la vitesse, voici les résultats :

durée totale des fichiers (en s)	durée execution de tout les fichiers (en s)	Ratio vitesse	Throughput estimé
240	55,35	4,33604336	156,097561
480	97,55	4,920553562	177,1399282
720	183,9	3,915171289	140,9461664
960	264,14	3,634436284	130,8397062
1200	301,51	3,979967497	143,2788299
		modèle : tiny	

durée totale des fichiers (en s)	durée execution de tout les fichiers (en s)	Ratio vitesse	Throughput estimé
240	97,04	2,473206925	89,0354493
480	176,23	2,723713329	98,05367985
720	317,53	2,267502283	81,6300822
960	378,22	2,538205277	91,37538998
1200	478,76	2,506475061	90,23310218
		modèle : base	

durée totale des fichiers (en s)	durée execution de tout les fichiers (en s)	Ratio vitesse	Throughput estimé
240	270,48	0,887311446	31,94321207
480	545,26	0,880313979	31,69130323
720	752,3	0,957065001	34,45434002
960	1311,25	0,732125834	26,35653003
1200	1420,1	0,845010915	30,42039293
		modèle : small	

durée totale des fichiers (en s)	durée execution de tout les fichiers (en s)	Ratio vitesse	Throughput estimé
240	778,76	0,308182238	11,09456058
480	1435,22	0,3344435	12,039966
720	2181,12	0,330105634	11,88380282
960	2861,25	0,335517693	12,07863696
1200	3630,68	0,330516597	11,89859751
		modèle : medium	

Au vu des temps d'exécution des modèles plus petits, le modèle large n'a même pas été mis en place (et également du fait que ma machine personnelle ne fait pas tourner le modèle large).

Pour ce qui est de la qualité de la retranscription, nous prendrons ici un exemple typique, mais l'entièreté du fichier a été comparé :

- Phrase originale : « En effet, seuls 33,35% des électeurs se sont déplacés le 10 avril, soit un taux d'abstention de 66,64%, alors que près d'un calédonien 1 sur 2 avait glissé un bulletin dans l'urne en 2017. Au premier tour, au final, la Calédonie est le territoire ultramarin où l'abstention aura été la plus forte. »

- Tiny : « Cela le 33,35 % des électeurs. Ce sont déplacées. Le 10 avril soit un taux d'absension de 66,64 % alors que près d'un qualé de néin sur 2, avait délicé un bulletin d'orlion en 2017. Au premier tour, au final, l'équalité de néin est le territoire Ultramarin, où l'absension aurait été la plus forte »

- Base : « En effet, seuls 33,35 % des électeurs se sont déplacés. Le 10 avril soit un taux d'abstention de 66,64 % alors que près d'un Calédonien sur deux avait glissé à un bulletin d'Orlion en 2017. Au premier tour, au final, la Calédonie et le territoire ultra-marin, où l'abstention aura été la plus forte. »

- Small : « En effet, seul 33,35% des électeurs se sont déplacés le 10 avril soit un taux d'abstention de 66,64% alors que près d'un calédonien 1 sur 2 avait glissé un bulletin d'Orléans en 2017. Au premier tour, au final, la Calédonie est le territoire ultramarin où l'abstention aurait été la plus forte. »

- Medium : « En effet, seuls 33,35% des électeurs se sont déplacés le 10 avril, soit un taux d'abstention de 66,64%, alors que près d'un calédonien 1 sur 2 avait glissé un bulletin dans l'urne en 2017. Au premier tour, au final, la Calédonie est le territoire ultramarin où l'abstention aura été la plus forte. »

Ces résultats nous permettent de conclure que l'utilisation du modèle large sera très compliquée à mettre en place, étant donné la durée de retranscription. Aussi, les modèles tiny et base donnent des résultats assez voire très éloignés de ce qui est originellement dit, leur usage devra autant que possible être écarté.

3. Encodage des fichiers

L'un des paramètres que nous nous devions de vérifier est l'encodage des différents fichiers audio. En effet, ce dernier a pour but de compresser au maximum la taille des fichiers, tout en essayant de préserver le maximum de qualité audio. Il existe 3 catégories d'encodages : autours de 124 kbps on parle de compression orienté espace de stockage où le but est de privilégier l'économie d'espace mémoire au prix d'une qualité audio moindre ; autours de 256 on parle de compression équilibrée, où l'audio est qualitatif, sans pour autant prendre trop de place ; la troisième catégorie a l'extension .FLAC ou .WAV et correspond à une qualité « lossless », c'est-à-dire que le fichier n'est pas du tout compressé, mais prends ainsi beaucoup plus de place.

La base de données utilisée pour nos précédents tests est encodée en 124 kbps, mais nous nous devons de vérifier l'impact qu'a l'encodage sur les durées de retranscription et sur le débit. A terme, la base « Hello World » est encodée en 256, ainsi pour nos projections nous aurons besoin de vérifier qu'elles ne seront pas différentes à case de l'encodage.

Le protocole de test que nous avons mis en place est donc de tester sur des fichiers de 4, 8, 12, 16 et 20 minutes, sur deux jeux de donnée, l'un encodé en 124 kbps et l'autre en 256 kbps. Le test est réalisé par le fichier **BasicTestWhisper.py**, avec le modèle **base** et sur la base de données **baseMod**. Voici les résultats :

durée totale des fichiers (en s)	durée execution de tout les fichiers (en s)	Ratio vitesse	Throughput estimé	Encodage		
240	75,23	3,190216669	114,8478001	124		
480	142,2	3,375527426	121,5189873	124		Moyenne ratio
720	235,11	3,062396325	110,2462677	124		3,21781381
960	301,04	3,188944991	114,8020197	124		
1200	366,75	3,27198364	117,791411	124		

durée totale des fichiers (en s)	durée execution de tout les fichiers (en s)	Ratio vitesse	Throughput estimé	Encodage		
240	74,86	3,205984504	115,4154422	256		
480	143,58	3,343083995	120,3510238	256		Moyenne ratio
720	226,17	3,18344608	114,6040589	256		3,204480486
960	317,97	3,01915275	108,689499	256		
1200	366,89	3,270735098	117,7464635	256		

Conclusion :

Comme nous le voyons, l'encodage ne représente aucunement un changement pour la retranscription par Whisper. En effet, selon nos analyses Whisper vectorise les fichiers avant de les traiter, ainsi la taille et l'encodage n'influence pas sur la durée de traitement. Pour la suite, les calculs prévisionnels réalisés sur les bases de tests pourront être extrapolés pour estimer les durées de traitement de la base réelle.

4. Transcription et multi-threading

Parmi les moyens que nous avons à notre disposition pour améliorer le débit, l'une des plus efficace n'est autre que le multi-threading. Alors qu'un programme classique exécute les instructions séquentiellement, en multi-threading on peut exécuter plusieurs séquences d'instruction en simultané. On appelle ces séquences des Threads et ces threads partagent les mêmes ressources mémoire tout en progressant indépendamment les uns des autres. Dans les faits, ce qui permet le multi-threading est le nombre de cœurs dans le CPU. On peut allouer un ou plusieurs cœurs à chaque processus, pour soit accélérer un processus (réduction du temps de réponse), soit pour paralléliser (augmentation du débit).

Comme expliqué plus tôt, les machines à notre disposition ont des CPU surpuissants, avec 36 cœurs pour la DELL 5820, et 2*40 pour la DELL PowerEdge T640. Nous avons ainsi un moyen de mettre en place le multi-threading à grande échelle.

Seulement voilà, python met en place un système qui empêche le multi-threading : le GIL (Global Interpreter Lock). Il s'agit d'un mécanisme de verrouillage qui empêche plusieurs threads d'exécuter du code python en simultanée. Il ne le fait pas par plaisir : python utilise un système de comptage de références pour gérer la mémoire et chaque objet en Python possède un compteur qui suit combien de références pointent vers lui. Ce compteur de références doit être protégé contre les conditions de concurrence, notamment dans le cas où deux threads voudraient essayer de changer sa valeur simultanément.

Le moyen que nous avons mis en place pour contourner ce problème de GIL se fait via le module Multiprocessing et plus particulièrement son outil Process. Cette méthode permet d'utiliser plusieurs processus plutôt que plusieurs threads, chacun avec son propre interpréteur Python et espace mémoire. A cela nous ajoutons le module Torch, qui nous permet d'allouer un espace sur le CPU pour effectuer des calculs particuliers. En pratique on divise ainsi un les taches à effectuer sur plusieurs processus, et à chaque processus on alloue un ou plusieurs cœurs.

Cependant, le nombre de cœurs alloués ne peut pas être le nombre total de cœurs de la machine. En effet, utiliser 100% des ressources disponible entrainerai une charge de travail telle

que nous nous exposerions à un risque de surchauffe et de blocage pour les services critiques. Nous n'utiliserons donc qu'au maximum 90% du nombre de cœurs total pour la suite des tests.

Ainsi, nous avons deux façons de mettre en place le multithreading : Augmenter le nombre de cœurs alloués à un même processus, ou augmenter le nombre de processus lancés en simultané. Nous avons ainsi lancé des tests sur ces deux façons de faire, de façon à comparer les débits offerts par chacune de ces solutions (ces tests sont lancés avec le modèle « base », sur la base de données **baseMtr,** sur la machine DELL 5820) :

1. **1 processus**, **N cœurs** : Nous commençons par tester l'impact de l'augmentation du nombre de cœurs alloués à un même processus via un programme python (« BasicTestWhisper.py »).

Durée totale des fichiers (en s)	Nombre de cœurs par fichier	durée execution de tous les fichiers (en s)	throughput
240	1	91,64	2,61894369
240	6	47,24	5,0804403
240	11	39,11	6,13653797
240	13	38,59	6,21922778
240	15	38,19	6,28436764
240	16	38,02	6,31246712
240	17	38,81	6,1839732
240	19	39,23	6,11776702
240	21	39,35	6,09911055
240	26	43,95	5,46075085
240	32	44,05	5,44835414

2. **N processus, 1 cœur** : En suite, nous allons augmenter le nombre de processus exécutés simultanément, toujours via le même programme python (« BasicTestWhisper.py »).

Durée totale des fichiers (en s)	Nombre de fichiers	Durée Min execution d'un fichier	Durée Max d'execution d'un fichier	throughput
240	1	92,15	92,15	2,604449267
1440	6	124,76	146,59	9,823316734
2640	11	136,53	160,92	16,40566741
3840	16	181,45	185,82	20,66515983
5040	21	223,4	227,7	22,13438735
5520	23	189,55	246,37	22,40532532
5760	24	203,81	254,44	22,63795001
6000	25	212,67	265,55	22,59461495
6240	26	201,5	276,23	22,58987076
6480	27	276,47	290,07	22,33943531
7680	32	286,12	373,18	20,57988102

Conclusion :

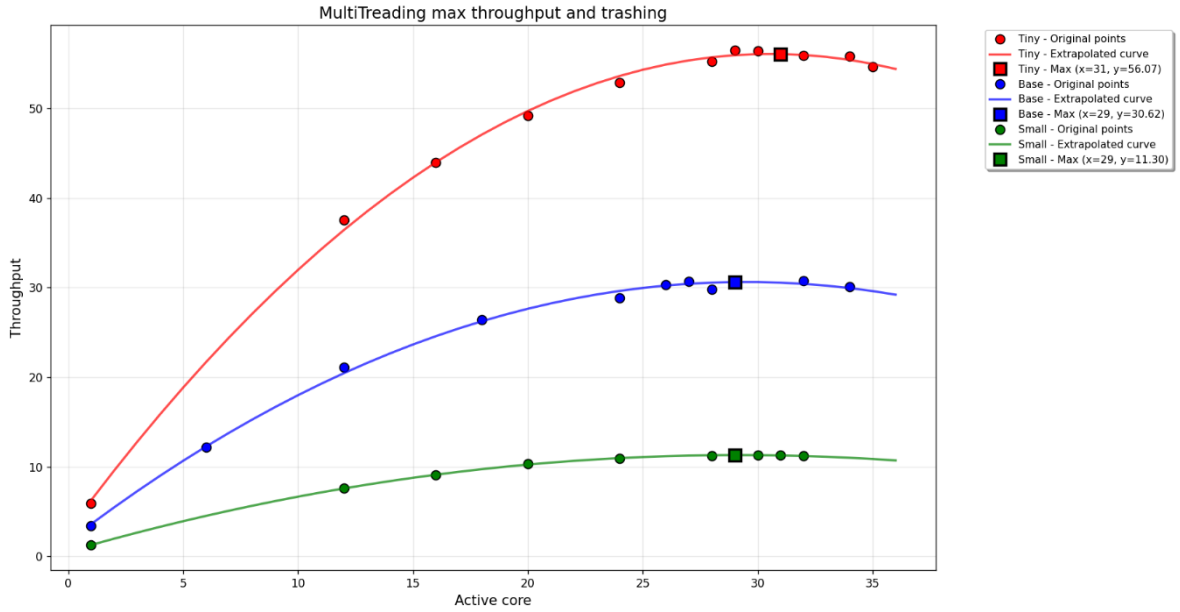
Nous voyons ici clairement ce qui a été expliqué plus tôt : L'augmentation du nombre de cœurs permet un gain en temps de réponse, c'est-à-dire que la transcription s'effectue plus rapidement. L'augmentation du nombre de processus en simultané permet l'augmentation du débit, c'est-à-dire que sur une période donnée le programme retranscrit plus de données.

Or le but de notre manœuvre est de traiter le plus de fichiers possibles pour une durée donnée. Notre choix se portera donc sur l'utilisation du plus de cœurs pour faire tourner des processus en simultanée.

5. Le cas du trashing

Si nous regardons les deux derniers tableaux, nous remarquons que l'augmentation du débit se fait de manière progressive, jusqu'à un point où le débit réduit. Nous attribuons cette diminution au fait que Whisper utiliser des threads en background, sans que nous n'ayons de moyens de lui interdire cet usage, ainsi lorsque les threads alloués aux processus de retranscription « débordent » sur le nombre de thread alloués à Whisper, Whisper tourne plus lentement et résulte de cela un ralentissement global de la vitesse de retranscription.

Nous avons besoin d'évaluer ce nombre de threads alloués automatiquement à Whisper, ainsi nous avons passé des tests sur chaque modèles (tiny, base, small, medium et large) afin de vérifier notre hypothèse, et d'avoir une idée précise du nombre de threads maximum à utiliser, afin d'éviter le ralentissement causé par le « trashing » (= surcharge) CPU. Les tests sont effectués avec la base de données **BaseMtr**, sur la machine « DELL 5820 ».



Pour les deux autres modèles, medium et large, nous observons un autre problème de trashing, non pas du CPU, mais de la mémoire vive. En effet, lorsque les modèles sont « chargés », ils sont montés en mémoire afin d'être utilisés par le CPU. Or dans certains cas que nous allons détailler, la montée en mémoire ne peut pas se faire par manque d'espace (bien que la machine soit équipée de deux barrettes de RAM de 32Go chacune). Ces points de trashing mémoire seront indiqués par « CRASH » dans les tableaux suivants :

5760	24	CRASH	CRASH	CRASH	CRASH	medium
4800	20	CRASH	CRASH	CRASH	CRASH	medium
3840	16	CRASH	CRASH	CRASH	CRASH	medium
3600	15	1253,88	1132,726	2,871088142	3,178173715	medium
3360	14	1165,1	1153,287143	2,883872629	2,913411478	medium
2880	12	1093,27	1079,635	2,634298938	2,667568206	medium

5760	24	CRASH	CRASH	CRASH	CRASH	large
4800	20	CRASH	CRASH	CRASH	CRASH	large
3840	16	CRASH	CRASH	CRASH	CRASH	large
3360	14	CRASH	CRASH	CRASH	CRASH	large
2400	10	CRASH	CRASH	CRASH	CRASH	large
2160	9	CRASH	CRASH	CRASH	CRASH	large
1920	8	1969,06	1945,09	0,975084558	0,987100854	large
1680	7	1814,98	1847,02	0,925630034	0,909573259	large
1440	6	1673,15	1664,408333	0,860652063	0,865172308	large

Conclusion :

Ces tests nous permettent de voir que les cas de trashing CPU arrivent toujours au même nombre de threads alloués, ce qui permet de conclure que Whisper utilise bel et bien des threads en background, bien qu'on ne les lui ait pas alloués. On estime ainsi le nombre de threads à allouer à notre multi-threading à 30. Il s'agit d'ailleurs du nombre où la performance est la plus forte en termes de débit.

Aussi, pour les modèles médium et large le problème n'est pas le trashing CPU mais le trashing mémoire. Whisper ne tourne juste pas lorsqu'on le charge dans plus de 15 processus pour medium, et 8 pour large. Ce problème pourrait être réglé (ou en tout cas repoussé) si nous disposions d'un moyen d'augmenter la capacité en mémoire vive de notre machine, or nous n'avons pas mieux que 64Go.

Ainsi, pour les tests à échelle que nous effectuerons plus tard, nous avons maintenant le nombre de threads optimisé pour chaque modèle :

Tiny : 31 - Base : 29 - Small : 29 - Medium : 15 - Large : 8

6. Transcription multi disque

Lors du processus de création des différents programmes de tests, la méthode est toujours la même, le programme est préparé et testé sur ma machine personnelle, puis mis en pratique sur la machine DELL 5820. Or par la force des choses nous nous sommes rendu compte que les durées d'exécution des retranscriptions étaient plus courtes sur ma machine personnelle que sur la DELL 5820. Cette observation est particulièrement surprenante lorsqu'on compare les deux CPU. Nous avons ainsi comparé les deux architectures grâce à un test effectué par le BasicWhisperTest.py sur la base de données **BaseMod**, sur les modèles tiny, base, small et medium :

Throughput moyen PC PERSO small	Throughput moyen DELL small
1,445351804	0,904845447

Nous tombons bien sur les des résultats confirmant nos observations. Or il existe deux explications possibles à ce changement : Soit le CPU de ma machine personnelle est plus rapide si on compare thread à thread, soit la différence de vitesse est due au fait que ma machine est équipée d'un SSD, permettant ainsi une lecture/écriture plus rapide au moment de la retranscription. Après recherche, le débit d'accès en lecture causé par notre utilisation serait de 1 à 2 Mo/s, en sachant que les disques en SAS de la machine DELL supportent jusqu'à 100 Mo/s. Cependant nous souhaitions vérifier cela lors d'une application sur la machine.

Un protocole a donc été mis en place : Nous allons comparer la lecture et l'écriture en Mono-disque, a celle en Multi-disques (sur les 3 disques disponibles sur la DELL). Afin de stresser au maximum les disques, nous allons utiliser le modèle le plus rapide (tiny), pour tester en conditions extrêmes l'impact qu'aurait le multi-disques.

En plus de cela, même si les tests ne sont pas concluants, nous espérons pouvoir gagner du temps sur la transcription effectuée sur plusieurs disques, ainsi les tests seront à double objectifs : vérifier si l'architecture de mémoire a un impact sur la durée de transcription, et vérifier s'il est possible de gagner du temps en répartissant la lecture et l'écriture sur plusieurs disques.

Les tests sont effectués de la manière suivante : Nous mettons en place du multi-threading avec 24 cœurs, chaque cœur est attribué à une liste de 24 fichiers d'une heure qu'il retranscrira de manière séquentielle (en batch, nous y reviendrons, ici il permet simplement de calculer une moyenne plus précise). Pour ce faire nous utiliserons le fichier bigTestWhisper.py, la base de données **baseBatch,** et le modèle tiny.

Pour le 1er test, les 24 listes de 24 fichiers sont sur le même disque, voici les résultats :

Cœur	Durée totale fichier (en s)	Durée transcription totale	Durée transcription max	Durée transcription mean	Throughput min (avec durée max)	Throughput mean (avec durée mean)
n°1	86400	42007,09	2213,54	1750,295417	39,03249998	49,36309561
n°2	86400	41574,77	2004,4	1732,282083	43,10516863	49,87640341
n°3	86400	40336,89	2039,84	1680,70375	42,35626324	51,40703708
n°4	86400	41004,07	1969,28	1708,502917	43,87390315	50,5705897
n°5	86400	41749,91	2012,01	1739,579583	42,94213249	49,66717294
n°6	86400	42124,65	2028,66	1755,19375	42,58968975	49,22533481
n°7	86400	42295,16	2212,52	1762,298333	39,05049446	49,02688629
n°8	86400	42055,47	1967,82	1752,31125	43,90645486	49,30630902
n°9	86400	42051,13	2046,12	1752,130417	42,22626239	49,31139781
n°10	86400	41998,8	2000,36	1749,95	43,1922254	49,37283922
n°11	86400	41439,88	2116,41	1726,661667	40,82384793	50,03875494
n°12	86400	40384,78	2023,98	1682,699167	42,68816886	51,34607642
n°13	86400	41807,91	2150,67	1741,99625	40,17352732	49,5982698
n°14	86400	40625,41	1968,55	1692,725417	43,89017297	51,04194641
n°15	86400	41267,72	1933,47	1719,488333	44,6864963	50,2475058
n°16	86400	41727,41	2175,37	1738,642083	39,71738141	49,69395417
n°17	86400	41566,01	2037,17	1731,917083	42,41177712	49,88691481
n°18	86400	41483,23	1992,47	1728,467917	43,36326268	49,98646441
n°19	86400	42105,15	2200,25	1754,38125	39,26826497	49,24813235
n°20	86400	41985,4	1995,84	1749,391667	43,29004329	49,38859699
n°21	86400	41652,32	2031,2	1735,513333	42,53643167	49,78354147
n°22	86400	42029,96	2284,02	1751,248333	37,82804003	49,33623539
n°23	86400	41114,26	1985,15	1713,094167	43,52315946	50,43505587
n°24	86400	41541,72	2022,94	1730,905	42,71011498	49,91608436
	Durée totale des fichiers	durée totale transcription	Moyenne de durée max	Moyenne de temps moyen	moyenne throughput min	moyenne throughput mean
	2073600	997929,1	2058,835	1732,515799	42,04940764	49,87810829

Pour le 2ème, les 24 listes de 24 fichiers sont réparties sur 3 disques par lots de 8 listes, voici les résultats :

Cœur	Durée totale fichier (en s)	Durée transcription totale	Durée transcription max	Durée transcription mean	Throughput min (avec durée max)	Throughput mean (avec durée mean)	Disque
n°1	86400	42456,81	1988,36	1769,03375	43,45289585	48,84022139	1
n°2	86400	42099,56	1952,14	1754,148333	44,25912076	49,25467155	1
n°3	86400	41702,06	2003,43	1737,585833	43,12603884	49,72416231	1
n°4	86400	41567,54	2065,62	1731,980833	41,82763529	49,8850786	1
n°5	86400	39453,35	2023,38	1643,889583	42,70082733	52,55827452	1
n°6	86400	41153,88	2108,83	1714,745	40,97058559	50,38650062	1
n°7	86400	42177,71	2018,66	1757,404583	42,80066975	49,16340882	1
n°8	86400	41717,31	2179,75	1738,22125	39,63757312	49,70598536	1
n°9	86400	41692,72	2028,76	1737,196667	42,58759045	49,73530151	2
n°10	86400	41855,76	2063,71	1743,99	41,8663475	49,54156847	2
n°11	86400	41865,56	1887,33	1744,398333	45,77895757	49,52997165	2
n°12	86400	42428,08	2122,17	1767,836667	40,71304372	48,87329335	2
n°13	86400	41254,15	1943,19	1718,922917	44,46297068	50,26403404	2
n°14	86400	40480,31	2054,81	1686,679583	42,04768324	51,22490416	2
n°15	86400	41928,93	2081,07	1747,03875	41,51710418	49,45511369	2
n°16	86400	41915,66	1959,57	1746,485833	44,09130575	49,47077059	2
n°17	86400	41985,65	2046,79	1749,402083	42,21243997	49,38830291	3
n°18	86400	42139,1	1985,72	1755,795833	43,51066616	49,20845486	3
n°19	86400	42657,28	2074,27	1777,386667	41,65320812	48,61069435	3
n°20	86400	42304,74	2136,56	1762,6975	40,43883626	49,01578405	3
n°21	86400	41912,76	2044,34	1746,365	42,26302865	49,47419354	3
n°22	86400	42070,85	2113,35	1752,952083	40,88295834	49,28828393	3
n°23	86400	42483,43	2057,37	1770,142917	41,99536301	48,80961824	3
n°24	86400	41563,72	1993,54	1731,821667	43,33998816	49,88966339	3
	Durée totale des fichiers	durée totale transcription	Moyenne de durée max	Moyenne de temps moyen	moyenne throughput min	moyenne throughput mean	
	2073600	1002866,92	2038,863333	1741,088403	42,42236826	49,63742733	

<u>Conclusion :</u>

Comme prévu, les résultats sont les mêmes pour les deux techniques, en mono-disque ou en multi-disques. Les débits des disques étant largement supérieurs aux débits dont la retranscription nécessite, aucun ralentissement n'est causé. Nous n'avons ainsi pas trouvé le gain espéré par la répartition sur plusieurs disques. Cependant, ces tests nous ont permis de comprendre l'influence qu'avait le type de mémoire sur les performances de la retranscription : Un SSD permet une lecture/écriture bien plus rapide, et cela pourrait être une piste d'amélioration à apporter à la machine DELL 5820. L'influence du SSD est telle que, bien que ma machine personnelle ait un CPU beaucoup moins puissant que celui de la DELL 5820, le temps de réponse est plus court. Or comme expliqué plus tôt, nous cherchons le moyen le plus efficace en termes de débit, et non en termes de temps de réponse. Nous resterons donc sur la DELL 5820 pour la suite des expérimentations.

7. Transcription en batch

Le dernier point sur lequel nous espérons un gain en temps, est la transcription en batch. Jusqu'à présent, nos tests n'étaient effectués que sur un fichier par thread, c'est-à-dire en traitement en temps réel. L'exécution en batch est un mode de traitement où un ensemble de taches et d'opérations sont exécutées de manière groupée. Concrètement dans notre cas, cela revient à créer une « file d'attente » pour chaque thread, et donc de traiter un nombre important de fichiers sans avoir à relancer le chargement des modèles en entier à chaque fois.

En effet, selon nos prédictions, bien que Whisper ai besoin de se relancer pour chaque retranscription, nous pensons qu'une partie de l'analyse est pipeliné. Cela permettrait un léger gain, de 5 à 15% de débit. Et comme expliqué plus tôt, 15% de vitesse sur une base de données de 588 heures, on pourrait gagner entre 29h et 88,2h. Cette valeur ne serait encore moins négligeable sur une base de données réelle, puisque les durées montent très rapidement.

Ce test revient à faire une expérimentation en conditions réelles, puisque la méthode batch permet le gain de temps, mais aussi le traitement automatisé d'un grand nombre de fichiers, ce qui permet de ne pas avoir à lancer le programme 10000 fois.

La méthodologie est simple : Nous avons précédemment calculé (dans la partie **5. Le cas du trashing**) le nombre de thread optimisé pour chaque modèle, nous allons ainsi lancer un test à grande échelle sur chaque modèle. Notre seule problématique est que la machine doit rester utilisable la journée, afin de permettre d'autres tests plus courts. Ainsi nous lancerons les tests la nuit, sur une plage horaire de 12h. Avec les données précédemment trouvées expérimentalement, nous allons donc adapter la base de données pour chaque modèle, afin que le traitement total ne dure pas plus de 12h :

modèle	tiny	base	small	medium	large
Temps exp souhaité (h)	12	12	12	12	12
Débit total	56	31	11,25	3,18	0,99
Thread	30	30	30	15	8
Débit / thread	1,87	1,03	0,38	0,21	0,12
Nombre de fichiers d'1h par thread	24	14	5	3	2

Afin de mesurer précisément l'utilisation du CPU, nous avons mis en place un monitoring en plus des tests de durées. Ce dernier nous permet de vérifier si le programme tourne bien toute la période durant laquelle elle a été traité, ainsi que de vérifier qu'aucun crash n'a ralenti l'exécution. Aussi, nous prendrons plusieurs mesures (un minimum de 5) afin de pouvoir extrapoler nos points, afin d'avoir un aperçu de ce à quoi ressemblerait le lancement avec plus de thread, impossible à réaliser pour l'instant à cause du trashing mémoire.

1er test : Modèle **tiny** - base **baseBatch – BigTestWhisperMonitored.py** :

cœur actifs	fichiers par cœu	durée totale a traiter par cœur	durée totale transcription	Throughput total	Throughput par cœur	% CPU théorique	% CPU moyen	difference theorique pratique CPU	% Mémoire moyen	Pics mémoire moyen	Plus haut pic mémoire
8	12	57600	3 027,91	19,02302248	2,37787781	22,22222222	28,47	6,247777778	21,98	22,35	22,8
16	12	115200	3 625,72	31,77299957	1,985812473	44,44444444	51,05	6,605555556	26,75	27,45	28,2
20	12	144000	4 037,38	35,66669474	1,783334737	55,55555556	62,35	6,794444444	28,96	29,57	30
22	12	158400	4 308,80	36,76197549	1,670998886	61,11111111	67,29	6,178888889	30,19	30,78	31,1
24	12	172800	4 605,72	37,51856387	1,563273495	66,66666667	74,13	7,463333333	31,23	31,96	32,7
26	12	187200	4 838,88	38,68663823	1,487947624	72,22222222	79,44	7,217777778	32,38	32,99	34,2
28	12	201600	5158,49	39,081204	1,395757286	77,77777778	84,66	6,882222222	33,54	34,22	39
30	12	216000	5504,83	39,23826894	1,307942298	83,33333333	90,02	6,686666667	34,68	35,39	36,3
								MODELE : TINY			

2ème test : Modèle **base** - base **baseBatch – BigTestWhisperMonitored.py** :

cœur actifs	fichiers par cœu	durée totale a traiter par cœur	durée totale transcription	Throughput total	Throughput par cœur	% CPU théorique	% CPU moyen	difference theorique pratique CPU	% Mémoire moyen	Pics mémoire moyen	Plus haut pic mémoire
8	8	38400	2 522,49	15,22305341	1,902881676	22,22222222	30,96	8,737777778	24,5	25	26,1
16	8	76800	2 996,98	25,62579664	1,60161229	44,44444444	52,18	7,735555556	31,1	31,8	33
20	8	96000	3 403,02	28,21023679	1,410511839	55,55555556	61,5	5,944444444	34,11	34,95	36,1
22	8	105600	3 653,96	28,90015216	1,31364328	61,11111111	67,92	6,808888889	35,69	36,46	37,7
24	8	115200	3 885,81	29,64632856	1,23526369	66,66666667	73,57	6,903333333	37,39	38,23	39,2
26	8	124800	4 133,12	30,19510684	1,161350263	72,22222222	79,06	6,837777778	38,9	39,76	41,8
28	8	134400	4412,98	30,45561049	1,087700375	77,77777778	83,68	5,902222222	40,59	41,69	46,1
30	8	144000	4709,11	30,57902661	1,019300887	83,33333333	89,27	5,936666667	42,2	43,05	45,6
								MODELE : BASE			

3ème test : Modèle **small** - base **baseBatch – BigTestWhisperMonitored.py** :

cœur actifs	fichiers par cœu	durée totale a traiter par cœur	durée totale transcription	Throughput total	Throughput par cœur	% CPU théorique	% CPU moyen	difference theorique pratique CPU	% Mémoire moyen	Pics mémoire moyen	Plus haut pic mémoire
8	3	14400	4 455,45	3,231996768	0,403999596	22,22222222	31,11	8,887777778	33,53	35,08	36,9
16	3	28800	5 511,85	5,225105908	0,326569119	44,44444444	52,21	7,765555556	47,94	49,96	52,3
20	3	36000	5 885,91	6,116301472	0,305815074	55,55555556	63,35	7,794444444	52,96	56,26	62,6
22	3	39600	6 251,07	6,334915462	0,287950703	61,11111111	68,72	7,608888889	57,83	60,31	67,1
24	3	43200	7 005,58	6,16651298	0,256938041	66,66666667	74,21	7,543333333	59,85	63,11	70,2
26	3	46800	7 409,80	6,315959945	0,242921536	72,22222222	79,43	7,207777778	63,97	66,92	76,7
28	3	50400	7765,96	6,48986088	0,231780746	77,77777778	84,63	6,852222222	67,15	71,02	78,8
30	3	54000	CRASH	CRASH	CRASH	83,33333333	CRASH	CRASH	CRASH	CRASH	80,9
								MODELE : SMALL			

4ème test : Modèle **small** - base **baseBatch – BigTestWhisperMonitored.py** :

cœur actifs	fichiers par cœu	durée totale a traiter par cœur	durée totale transcription	Throughput total	Throughput par cœur	% CPU théorique	% CPU moyen	difference theorique pratique CPU	% Mémoire moyen	Pics mémoire moyen	Plus haut pic mémoire
1	2	1200	2 692,11	0,445747016	0,445747016	2,777777778	8,42	5,642222222	19,46	21,58	25,9
8	2	9600	4 840,34	1,983331749	0,247916469	22,22222222	29,29	7,067777778	54,43	59,2	77,8
10	2	12000	5 073,36	2,365296372	0,236529637	27,77777778	34,44	6,662222222	64,13	73,13	86,3
12	2	14400	5 326,44	2,703494266	0,225291189	33,33333333	41,33	7,996666667	72,64	80,32	99,9
13	2	15600	5 456,69	2,858875985	0,219913537	36,11111111	43,16	7,048888889	74,61	85,82	99,2
14	2	16800	5 657,16	2,969687971	0,212120569	38,88888889	47,04	8,151111111	79,16	90,28	100
15	2	18000	CRASH	CRASH	CRASH	CRASH	CRASH	CRASH	CRASH	CRASH	100
16	2	19200	CRASH	CRASH	CRASH	CRASH	CRASH	CRASH	CRASH	CRASH	100
								MODELE : MEDIUM			

5ème test : Modèle **small** - base **baseBatch – BigTestWhisperMonitored.py** :

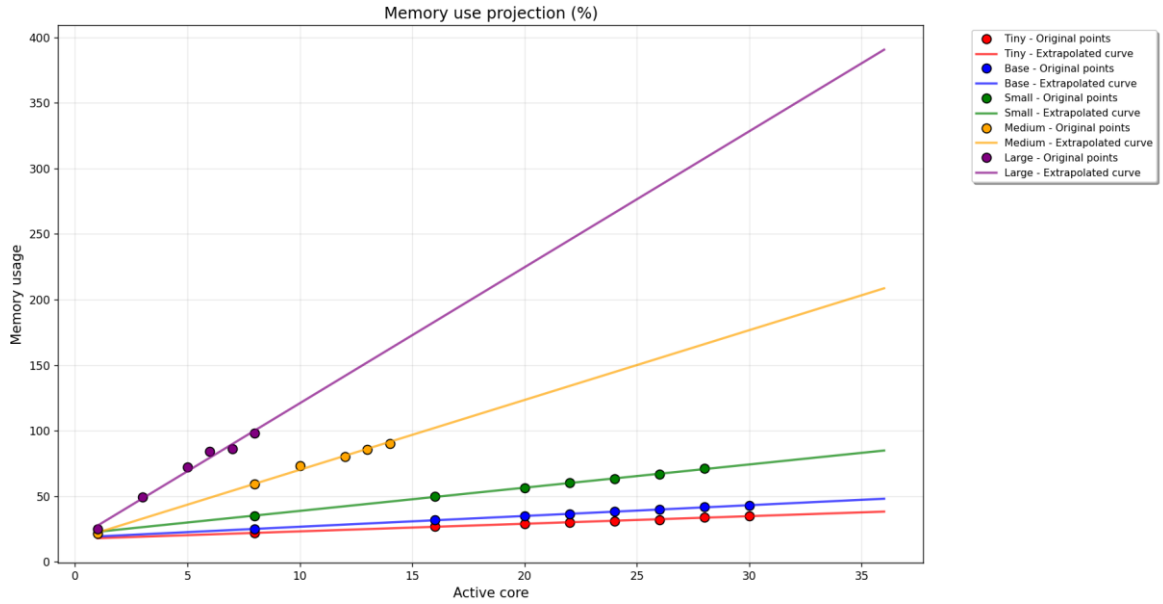
cœur actifs	fichiers par cœu	durée totale a traiter par cœur	durée totale transcription	Throughput total	Throughput par cœur	% CPU théorique	% CPU moyen	difference theorique pratique CPU	% Mémoire moyen	Pics mémoire moyen	Plus haut pic mémoire
1	1	600	2 354,83	0,254795463	0,254795463	2,777777778	8,47	5,692222222	22,07	24,77	25,7
3	1	1800	3 864,20	0,465814399	0,155271466	8,333333333	13,2	4,866666667	39,01	49,26	51,9
5	1	3000	4 009,59	0,748206176	0,149641235	13,88888889	19,22	5,331111111	58,15	72,07	77,4
6	1	3600	4 131,62	0,871328922	0,145221487	16,66666667	23,3	6,633333333	71,76	84,08	94,1
7	1	4200	4 220,13	0,995230005	0,142175715	19,44444444	25,45	6,005555556	74,55	86,19	100
8	1	4800	4 449,63	1,078741378	0,134842672	22,22222222	29,44	7,217777778	87,72	97,93	100
9	1	5400	CRASH	CRASH	CRASH	CRASH	CRASH	CRASH	CRASH	CRASH	100
10	1	6000	CRASH	CRASH	CRASH	CRASH	CRASH	CRASH	CRASH	CRASH	100
								MODELE : LARGE			

Conclusion :

Avec ces tests nous remarquons que le lancement en batch ne permet pas d'augmenter notre débit, voir même parfois nous remarquons un ralentissement. Malheureusement le traitement

par batch est obligatoire pour traiter les données que nous visons, ainsi ce ralentissement sera présent dans la suite de nos tests, sans que nous ne puissions le contourner.

Afin de poursuivre les tests, la machine DELL devra être améliorée, et équipée avec plus de mémoire vive, comme le montre les projections suivantes :



On voit que si nous souhaitons lancer les tests sur les modèles medium et large, nous devons nous équiper en mémoire respectivement à 200% et a 350% de la mémoire que nous possédons actuellement, c'est-à-dire 128Go et 224Go. Cette augmentation de RAM fera l'objet d'une demande auprès des services concernés (nous y reviendrons dans la partie 4).

IV. Pistes d'amélioration

Il existe encore des pistes d'améliorations à explorer, faute de temps. En effet que ce soit en efficacité, en coûts énergétiques ou même en terme qualitatifs ; nous n'avons pu tester l'entièreté des idées que nous avions.

Ces dernières sont séparées en deux catégories : Les améliorations logicielles, et les améliorations matérielles.

1. Améliorations logicielles

Premièrement, notre outil est entièrement basé sur une solution qui contourne le GIL de python comme expliqué dans la partie III.4. Or cette solution ne correspond pas réellement à du multi-threading. Il s'agit plutôt d'une simulation de multi-threading, en effet le multi-threading permet à des threads de travailler sur les mêmes ressources, en en partageant certaines et dans notre solution chaque thread est complètement isolé des autres. Une solution via un autre langage que python pourrait permettre de mettre en place ce « vrai » multi threading.

Ensuite, le problème de surcharge RAM expliqué dans la partie III.7 provient du fait qu'au lancement Whisper fait monter en mémoire l'entièreté des données qu'il a traiter (cad les x fichiers audios). Un moyen qui pourrait nous permettre d'éviter le crash serait de créer une file d'attente de monté en mémoire des données. Admettons avoir besoin de 10Go, mais n'en possédant que 8Go. Alors nous pourrions charger les 8 premiers Go, et dès lors que la mémoire se décharge, monter les 2Go manquant. Cette solution est réaliste mais chronophage. En effet il faudrait tester pour comprendre en détail comment fonctionne Whisper, et mettre en place une solution optimisée de file d'attente, et durant ce stage nous n'en avions pas le temps.

Enfin, nous possédons deux machines (la DELL PowerEdge T640 et la DELL 5820) dont l'une des deux a été mise complétement de coté durant mon stage. La raison principale de cet écartement a été la difficulté que j'ai rencontrée pour mettre en place du multi-threading sur cette machine. En effet l'outil que nous avons utilisé tout du long a été le module multi-threading du package Process, or ce dernier n'est pas conçu pour fonctionner en bi-processing. Et le développement en bi-processing étant relativement moins populaire en termes de documentation sur internet, nous avons décidé de mettre de coté ce pan de notre recherche d'amélioration du débit. Or en estimant le plus raisonnablement (donc sans compter sur le fait que les cœurs des deux CPU de la DELL PowerEdge T640 sont plus puissants que ceux sur la machine que nous avons utilisée), nous devrions multiplier notre débit par ~2,5. Et ce donc uniquement par le fait que le nb de cœurs est ~2,5 fois plus élevé. Il s'agit donc de la priorité dans les solutions à explorer pour les stagiaires suivants.

2. Améliorations matérielles

Le moyen le plus simple que nous avons de gagner du débit, serait d'améliorer les composants de notre machine. Ainsi, comme montré dans les tests, 3 parties sont essentielles pour l'efficacité du programme, ainsi l'amélioration de ces derniers fera forcément gagner du temps :

Premièrement, l'augmentation de la RAM. En effet, comme expliqué dans la partie III.7 le ralentissement vient du fait que nous ne pouvons pas lancer trop de threads en même temps, a cause du manque de ressources RAM nécessaire a la monté en mémoire des fichiers à analyser par Whisper. Ainsi si nous souhaitons lancer plus de threads en simultanée, il faut augmenter la RAM de la machine, afin d'atteindre l'équilibre entre puissance CPU et mémoire. Ainsi au vu des graphs d'extrapolation de le partie III.7, pour faire tourner tous les modèles en 30 threads avec le modèle Large, nous aurions besoin de 256Go de RAM.

Ensuite, le moyen le plus simple d'augmenter le débit serait d'augmenter le nombre de thread, donc le nombre de cœurs, et donc de remplacer le CPU par un CPU plus puissant. Cette solution est simple mais extrêmement couteuse, ainsi elle ne sera pas explorée du tout, mais dans l'absolu reste une solution pour augmenter le débit.

Enfin, comme expliqué dans la partie III.6, le point qui fait qu'a nombre de thread égal, ma machine personnelle est plus rapide que la DELL 5820 est le fait que ma machine est équipée d'un SSD. Ainsi la vitesse de montée en mémoire et celle d'écriture est augmentée, permettant donc de gagner du temps d'exécution. La machine DELL 5820 n'est pas équipée de SSD car elle sert habituellement de machine de capture, et il est ainsi plus logique d'avoir privilégié une grande capacité de stockage a une grande vitesse de lecture/écriture. Or dans notre cas, le SSD nous ferait gagner du temps, ainsi pour ce projet il pourrait être intéressant d'explorer cette piste.

V. Applications

1. Préparation du programme de transcription

Maintenant que l'entièreté des tests ont été réalisés, nous connaissons les combinaisons de réglage des paramètres optimaux. Ainsi nous sommes en mesure de produire des retranscriptions avec une vitesse satisfaisante sur des base de données de grande taille. L'objectif suivant est donc de créer un programme qui nous permettra de lancer la retranscription d'une base quelconque.

Pour cette partie, nous avons un nouveau problème auquel nous n'avions pas encore fait face : notre base n'est pas constituée de données à durées homogènes. En effet nos captures correspondent à des émissions télévisuelles, et il n'y a pas de normes en termes de longueur pour ces dernières. Ainsi la répartition de nos fichiers est très large, allant de 600s pour les plus petit jusqu'à 18000s pour le plus grand.

Ainsi nous avons dû trouver une solution afin de répartir équitablement la charge de travail pour chaque thread. Le programme « Repartition.py » correspond à ce programme de répartition. Via un algorithme glouton (greedy en anglais), le but est de minimiser l'écart entre la durée moyenne de chaque batch et la durée maximum des batch ; avec notre configuration et sur la base de données « helloWorld » cette différence est en dessous de 0,5%, ce qui est très largement convenable comme résultat.

Un deuxième programme a dû être réalisé en amont : « Parse.py ». Il sert d'adapter pour rendre lisible la base de données algorithmiquement. Son fonctionnement est simple : nous lui donnons un path ainsi qu'un suffixe de fichier (ex : audio.mp4, .mp3 ...), et il va parcourir tous les dossiers et sous dossiers présents dans le path et noter dans un csv le path de tous les fichiers possédant le suffixe renseigné, ainsi que leur durée. Cela permettra à « Repartition.py » de créer des objets avec l'attribut « path » et l'attribut « durée », afin de mettre en place plus rapidement et simplement notre algorithme de répartition, mais aussi cela donnera la possibilité à l'utilisateur de vérifier que le csv contient bien l'entièreté des fichiers souhaités.

Et enfin, le programme principal « OperationnalWhisperTranscriptor.py » lance la retranscription et le monitoring CPU et mémoire de cette dernière. Pour chaque fichier présent dans le fichier csv produit par « Parse.py », le programme créé un fichier txt contenant la transcription effectuée par le modèle sélectionné.

2. Horodatage

Après la présentation de notre outil aux différentes équipes de recherche, une demande nous a expressément été faite : ajouter un outil d'horodatage à nos retranscriptions.

En effet, pour rappel l'équipe pour laquelle ce projet est prévu cherche à repérer les entités nommées parmi des audios, et afin de gagner du temps dans la recherche de certains fichiers, ajouter un système d'horodatage permettrai d'économiser le temps et l'énergie de cette recherche. Ainsi nous avons ajouté cette fonctionnalité, via un module natif à Whisper appelé « segment », ce qui nous permet de passer d'une retranscription de ce format :

de ce deuxième tour et la même copromie est les calidonniens vontiles à l'évoiller. En effet, seulement 33,35 % des électeurs se sont déplacées. Le 10 avril soit un taux d'absension de 66,74 %

A ce format :

00:00:00,000 --> 00:00:03,620

de ce deuxième tour et la même copromie est

00:00:03,620 --> 00:00:05,860

les calidonniens vontiles à l'évoiller.

00:00:06,080 --> 00:00:10,099

En effet, seulement 33,35 % des électeurs se sont déplacées.

00:00:10,359 --> 00:00:15,300

Le 10 avril soit un taux d'absension de 66,74 %

3. Amélioration du programme

À la suite de la mise en commun du trio de programme précédemment présentés, un problème nous a été mentionné a de nombreuses reprises : la prise en main est trop compliquée pour un utilisateur non habitué à l'utiliser.

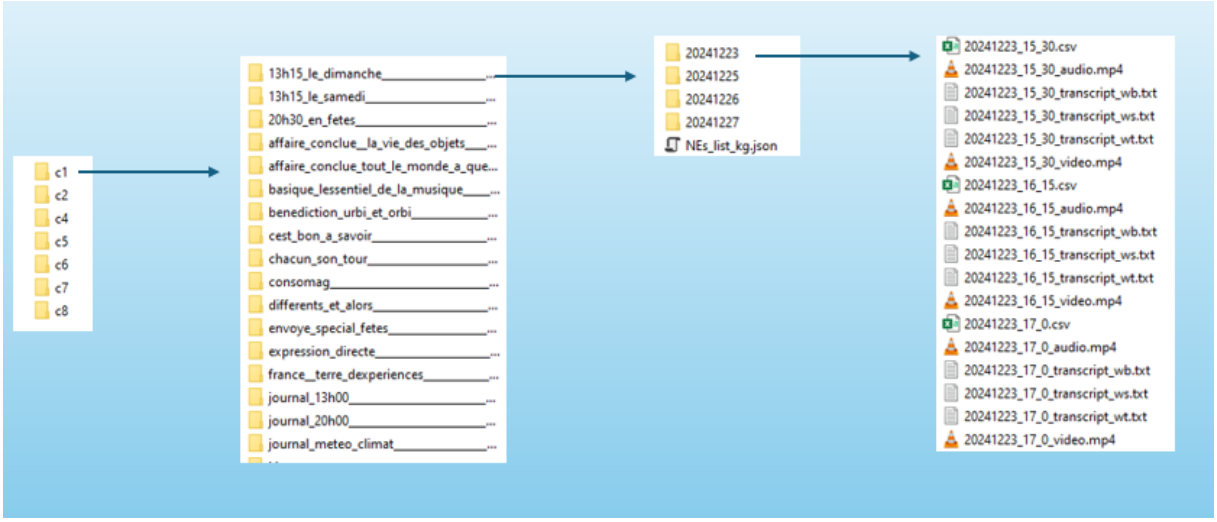
Ainsi, les programmes ont été remaniés et fusionnés, afin de n'avoir à régler les paramètres dans un seul des programmes (appWhisper.py). Une notice d'utilisation a également été rédigée, afin de faciliter la prise en main du programme.

4. Architecture de la base Hello

Comme expliqué au début de rapport, la tache directrice du stage était la préparation et la retranscription de la base de données Hello world. Cette dernière se présente sous la forme suivante :

Contenu : Voir II.3

Format :



Comme présenté dans le schéma, les captures sont triées dans des dossiers par date, eux même par émissions. Chaque émission est classée dans le fichier c correspondant à la chaine de capture.

5. **Transcription de la base Hello**

Comme expliqué précédemment, la demande de l'équipe avec laquelle nous travaillons nous a chargé de produire la transcription dans les deux formats (avec et sans horodatage), afin de pouvoir mener leurs propres expérimentations.

Nous avons donc lancé la retranscription à grande échelle sur les modèles tiny, base et small (les autres modèles étant trop lents, ils n'ont pas pu être conduits sur la durée du stage).

Voici le tableau récapitulatif :

Model	Tiny	Base	Small
Files	849		
Transcription rate	100%		
Process time	19.9h	31.4	79.02
Throughput	41	31	12

La base a donc pu être retranscrit sans problème ni crash, et dans sa totalité.

6. **Publication de la base hello**

Avant la publication de la base, il a été nécessaire de la « nettoyer ». En effet lors de l'analyse de la base pour préparer le script « Parse.py », nous avons remarqué que certains fichiers étaient de durée nulle (correspondant à un fichier corrompu), il a donc fallu lister et supprimer tout les fichiers vides. Aussi, lors de la création des différentes package d'émissions, certaines ont été dupliquée, ou séparées alors qu'elles étaient les mêmes. Par exemple « paris_2024_lemotion_des_jeux » correspond à la même émission que « paris_lemotion_des_jeux ». Ainsi il a fallu regrouper les différentes émissions aux noms similaires qui ont été séparées à tort.

Maintenant que la base est prête, nous devons la publier. Simplement, la base de données faisant 1,12To nous devions la compresser, et dans le but de permettre un téléchargement plus pratique nous avons fragmenté l'archive. Notre choix de fragmentation s'est porté sur un découpage en 16 x 2024^3 (soit 17 179 869 184 octets par package).

La publication de la base a été effectuée sur le site https://dataset-stvd.univ-tours.fr/mner/ avec une taille compressée de 281Go, donc repartis en 16 fichiers à télécharger séparément.

La publication fragmentée permet d'abord de stabiliser le téléchargement, en particulier pour une partie de l'équipe qui est actuellement au Vietnam, où la connexion internet nécessite une attention particulière à l'intégrité des téléchargements du fait de son instabilité.

VI. Taches annexes

En parallèle des activités menées durant le stage, il m'a été confié plusieurs taches spécifiques en lien avec l'activité première.

1. Réception et étiquetage de disques durs

Pour le projet Station TV, le RFAI possède un lot de disques durs professionnels servant plusieurs buts. La logique actuelle du laboratoire est de toujours posséder le double de la capacité nécessaire, afin de garantir un backup complet des données en cas de bug.

Parmi ces disques, la première catégorie est les disques de « backup » des différentes bases de données de capture. Cette catégorie est composée de 4 disques LaCie d2 Pro 20 To (https://www.ldlc.pro/fiche/PB00502054.html) pour une capacité totale de 80To donc actuellement 23,2% de la capacité est occupée. On y stock les bases « PVCD dataset », « Fact-Checking », « social TV dataset », « NER hello root », « NER hello collection » et « XMLTV ».

Le second groupe de disques est lui constitué de disques « portables » de prêt. En effet lors de certaines tâches, les chercheurs et doctorants peuvent avoir besoin d'augmenter leur capacité de stockage temporairement. Ainsi 5 disques sont chacun attribué à un membre de l'équipe :

- disque WD XBOX 12 To -> V.H Le pour « PVCD »
- disque WD XBOX 12 To -> F. Rayar pour « Fact-Check »
- disque WD XBOX 12 To -> LIFAT pour « VQA »
- disque Seagate Exos X16 16To -> G. Vu pour « MNER »
- disque LaCie d2 Pro 14 To -> T. Bourdeau pour « STT »

Dans le cadre du travail de G. Vu, une nouvelle capture plus étendue de la base de données NER est prévue, ainsi le besoin en espace de stockage va augmenter considérablement. Il a été convenu d'engager l'achat de disques de stockage HDD sur la thèse de G. Vu (sachant que les disques HDD02 et 06 lui ont été alloués pour les besoins de la thèse). L'objectif était l'achat à minima d'un disque de backup (20 To) et de travail (12-14 To) pour remplacement. Une demande de financement a donc été créée et un accord de principe a été retourné sur budget de la thèse CIFRE MACIF (en collaboration RFAI / BDTLN). Il a été convenu d'engager l'achat à partir de fin février (à la réouverture de SIFAC). Pour normalisation, il a été convenu de basculer la commande en disques LaCie d2 14 (https://www.ldlc.pro/fiche/PB00352169.html) et 20 (https://www.ldlc.pro/fiche/PB00502054.html). Les disques 14 et 20 To ont été ciblés en termes de critère ratio prix / capacité et règle >500€ HT.

Après discussion avec les différents services concernés, un accord a été conclu sur l'achat en investissement de deux disques LaCie d2 Professional USB 3.1 20 To.

A la réception, il a ainsi fallu l'enregistrer dans les fichiers du bureau de Polytech, l'étiqueter avec un code servant à le tracer pour l'université, et un autre pour le suivre au niveau du LIFAT. Ensuite nous avons testé le disque, en lecture et en écriture via un petit script python.

Avec l'achat de ces disques, la capacité de stockage du LIFAT a été portée à 186 To, dont 181,64To effectifs.

2. Préparation et extrapolation des graphs

A plusieurs reprises durant le stage, il a été nécessaire de trouver mathématiquement des points dans des courbes, où même d'extrapoler des points pour tracer ces mêmes courbes.

Pour ce faire, il a été convenu d'utiliser des librairies python déjà existantes, telles que numpy pour l'extrapolation et la recherche de point maximums, et matplotlib pour le traçage des différents graphiques.

Extrapolation :

Le code principal cette partie est le suivant :

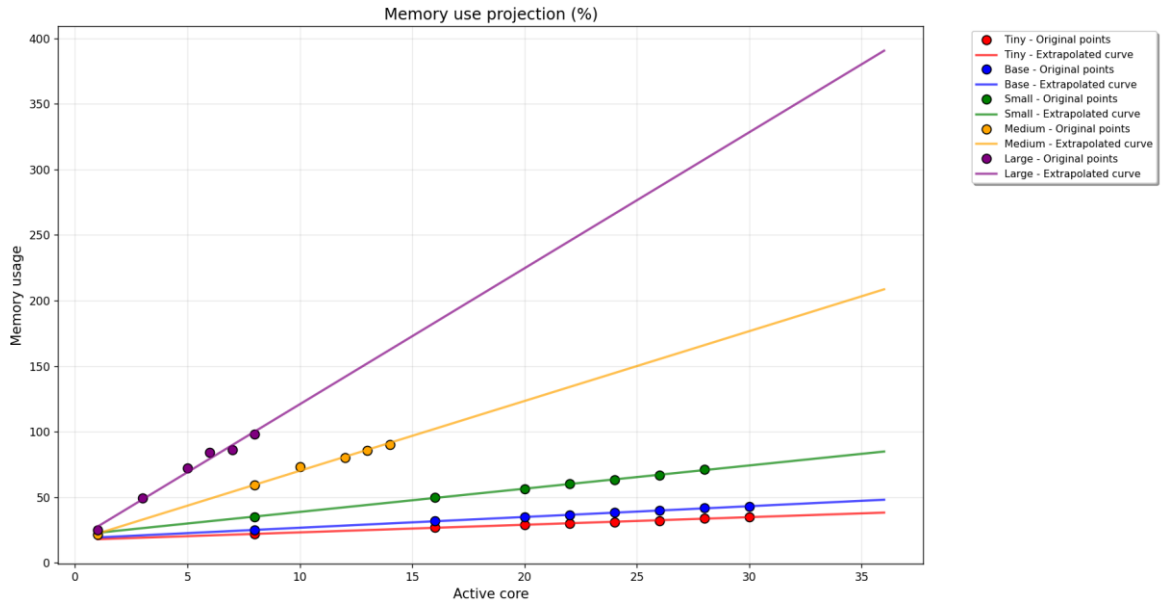
```python
def extrapoler_point(liste, n, degre=2):
    x = np.array([point[0] for point in liste])
    y = np.array([point[1] for point in liste])

    coefficients = np.polyfit(x, y, degre)
    polynome = np.poly1d(coefficients)
    return float(polynome(n))
```

L'usage de cette librairie nous a évité d'avoir à développer tout un outil permettant d'extrapoler les points trouvés, nous permettant de gagner un temps considérable.

Le fonctionnement est simple : à partir de points que nous fournissons, le programme trace la courbe de la fonction extrapolée. Pour avoir un résultat réaliste, il faut entre 5 et 8 points au minimum.

Ainsi dans le graph suivant, nous avons pu tracer les courbes d'usage de mémoire, en ne partant que des points marqués sur le graphique :



Traçage des graphiques :

La librairie que nous avons choisie a été matplotlib, pour son utilisation et sa prise en main simple.

3. Demande d'achat de RAM

Lors des tests présentés dans la partie III.7 nous avons conclu que la capacité actuelle en mémoire rapide de la machine Dell 5820 était notre facteur limitant. Pour rappel nous avions la capacité de ne faire tourner en 30thread que les deux plus petits modèles de Whisper. Lors de la montée en mémoire des fichiers à analyser par Whisper, un message d'erreur nous indiquait que la capacité RAM était trop faible par rapport au besoin de la tâche.

Ainsi après discussion avec M.Delalandre, il a été convenu de lancer une demande d'achat en investissement de barrettes de RAM.

La machine en 2022 avait été configurée en 4x 16 Go de RAM compte-tenu des 4 "memory channels" du CPU W-2295. La machine dispose de 8 slots RAM, néanmoins compte-tenu de la perte de performance en mapping 2 RAM -> 1 Channel et du différentiel de fréquence RAM 4 x16 Type 1 + 4 x16 Type 2, il a été recommandé un passage en 4 x 32 Go. En effet, nous avions 4 configurations possibles :

a. 128 Go (4 x 32)
b. 256 Go (8 x 32)
c. 256 Go (4 x 64)
d. 512 Go (8 x 64)

D'après nos projections, si nous souhaitons couvrir tous les modèles de Whisper (tiny, base, small, medium et large) nous devrions opter pour 256Go, ma les prix grimpent très vite et pour les configurations b,c et d nous serions sur un coût supérieur à 1000€.

En plus de cela, la machine ayant un usage professionnel et sensible, elle a été équipée de RAM ECC ("Error-Correcting Code"), faisant augmenter encore le prix des barrettes de RAM.

Le stage s'étant terminé en pleine préparation des devis d'achat de RAM, il ne m'a pas été confié de m'occuper de passer commande, ni de les réceptionner.

Capture audio large échelle et transcription automatique pour la Station TV

Auteur : Tom BOURDEAU (22205482)
Tuteur : Mathieu DELALANDRE
Période : 23 avril - 27 juin 2025
Laboratoire : LIFAT - Équipe RFAI (Reconnaissance des Formes et Analyse d'Images)
Université : Polytech Tours

Ce rapport présente les travaux menés sur l'optimisation de la transcription automatique de flux télévisuels dans le cadre du projet Station TV du LIFAT. L'objectif principal consistait à développer et optimiser un système de retranscription audio-texte en temps réel pour des contenus de la TNT, en utilisant l'outil Whisper d'OpenAI. Les recherches se sont concentrées sur l'amélioration des performances de traitement grâce au multi-threading et à l'optimisation des ressources matérielles disponibles (machines DELL PowerEdge T640 et DELL 5820). Différents modèles Whisper ont été évalués en termes de vitesse et de qualité de transcription. Les tests ont permis d'identifier les configurations optimales pour traiter efficacement des volumes de données importants (588 heures de contenu audiovisuel). Le projet a abouti à la création d'outils opérationnels de transcription avec horodatage et à la publication de la base de données "Hello World" pour la recherche en reconnaissance d'entités nommées.