Sorbonne Université
Master ANDROIDE

DISCO: Contextually Improving Command Discoverability

Stage de Master 1

Réalisé par :

Alexandre XIA

Encadré par :
Gilles BAILLY, ISIR, Sorbonne Université
Julien GORI, ISIR, Sorbonne Université

Référent :
Thibaut Lust, LIP6, Sorbonne Université

3 juillet 2023

Table des matières

Chapitre 1

Introduction

Les applications telles que PowerPoint ou Adobe Photoshop qu'on utilise tous les jours fournissent de nombreuses commandes selon nos besoins. Parmis ces commandes, certaines sont plus efficaces en regroupant d'autres commandes plus simples, ce qui permet d'accélérer une tâche. Par exemple, Adobe Photoshop propose la commande "retirer les yeux rouges" qui permet à l'usager de cliquer à partir de l'outil sur les zones correspondantes et de les repreindre en noir. Un certain nombre d'utilisateurs ne connaisse pas l'existence de cette commande et choisisse l'outil "pinceau" et changer sa couleur en noir avant de repeindre manuellement les zones rouges. Un autre exemple est l'alignement d'éléments sur Microsoft Word, l'usager qui ne connait pas la commande d'alignement procèdera par déplacer chaque objet un à un vers la position qui le convient alors qu'il aurait pu juste sélectionner les éléments et appeler la fonction d'alignement. Le problème étant que cela prenne plus de temps à exécuter

Plusieurs méthodes pour recommander des commandes ont été proposées mais une partie suggère que l'utilisateur est déjà familier avec les commandes optimales. Les outils permettant à l'utilisateur de découvrir de nouvelles commandes utilisent en majorité des recommandations selon ce que l'utilisateur fait.

Dans la continuité du projet de Master I, nous avons durant le stage essayer d'aboutir à une preuve de concept pour la création d'un modèle de recommandation de commandes. Ce modèle se baserait plus sur l'intention de l'utilisateur en utilisant l'état de l'application (une image ou un vecteur de descripteurs avant et après modification par l'utilisateur) plutôt que les commandes utilisées pour atteindre son objectif. Cette état serait donnée à un classifieur (appris ou non) qui pourra suggérer potentiellement la bonne commande. Le système est nommé selon l'article de nos encadrants "DISCO".

Pour cela, on cherche à tester des commandes pertinentes et pour lesquelles on a des effets assez différenciables. Par exemple, "ctrl+a" sélectionne tout le document, mais "ctrl+shift+fin" a le même effet si on se trouve au début du document ce qui rendrait difficile de savoir s'il faudrait suggérer "ctrl+a" qui serait la bonne commande si l'utilisateur sélectionne tout le document avec "shift" et les flèches.

Ce projet est fait en collaboration avec mes camarades : Christian ZHUANG et Nassim AHMED-ALI. Nous sommes encadré par Gilles BAILLY et Julien GORI.

Le lien du dépôt git est ici : NOTRE GITHUB

Chapitre 2

État de l'art

2.1 4 domains of interface performance improvement

En ce qui concerne le concept d'aide pour les commandes, il existe différentes approches permettant d'améliorer les performances de l'utilisateur avec les interfaces. Ces approches se basent sur 4 points :

- L'amélioration intramodale (intramodal improvement) concerne la vitesse et l'ampleur des performances d'une méthode spécifique du logiciel. Par exemple, aider l'utilisateur à prendre en main un clavier spécifique (ShapeWriter [1])

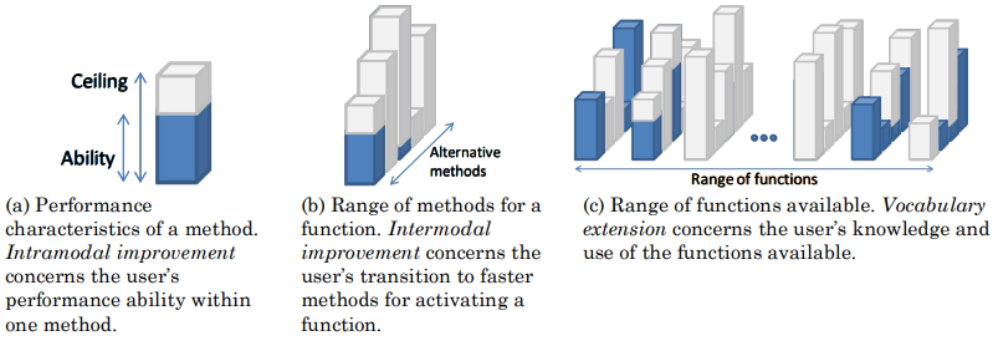
- L'amélioration intermodale (intermodal improvement) pour le passage vers des méthodes d'accès plus efficace pour une fonction spécifique avec un plafond ("ceiling") de performances plus élevé. Par exemple, Le passage d'une commande sélectionnée à la souris vers l'équivalent en raccourci clavier.

- L'extension vocabulaire (vocabulary extension) par rapport aux connaissances de l'utilisateur vis-à-vis des outils du logiciel. On voudrait que l'usager découvre et apprend un certain nombre de commandes d'un logiciel qui lui serait utile. Par exemple, l'utilisateur qui utilise habituellement "copier-coller" pourrait utiliser la commande "dupliquer" qui est une alternative (plus rapide dans ce cas).

Il faut noter que l'amélioration intermodale concerne une fonction spécifique et une alternative pour l'utiliser alors que l'extension vocabulaire concerne les commandes que l'utilisateur connaît.

- La planification des tâches (tasks mapping) qui concerne les stratégies employées par l'utilisateur pour faire une tâche (plus difficile). Pour dessiner 2 rectangles, il y aurait l'utilisateur qui en dessinerait 2 et un autre qui dupliquerait un rectangle qu'il aurait dessiné.

Ces notions sont toutes précisées dans l'article Supporting Novice to Expert Transitions in User Interfaces [1]



(a) Performance characteristics of a method. *Intramodal improvement* concerns the user's performance ability within one method.

(b) Range of methods for a function. *Intermodal improvement* concerns the user's transition to faster methods for activating a function.

(c) Range of functions available. *Vocabulary extension* concerns the user's knowledge and use of the functions available.

FIGURE 2.1 – Représentation des approches amélioration intramodale (a), amélioration intermodale (b) et de l'extension vocabulaire (c) [1]

2.2 2 Exemples d'approches pour l'extension vocabulaire (vocabulary extension) en lien avec notre projet

On s'intéresse de notre point de vue, plutôt à la notion d'extension vocabulaire. En effet, l'utilisateur n'a probablement pas conscience de l'existence des commandes et notre objectif est de pouvoir lui donner la possibilité de les découvrir.

2.2.1 Exemple : "SimCURL : Simple Contrastive User Representation Learning from Command Sequences"

L'article "SimCURL : Simple Contrastive User Representation Learning from Command Sequences" propose un système qui utiliserait un autre moyen que la prédiction classique qui nécessite une base de données entièrement labellisée. Les auteurs de l'article présente un problème comme quoi, les données ne sont en générales sans label (classe) et qu'en attribuer et coûteux financièrement et en temps. De plus, l'interprétation pour les classifier manuellement nécessite une bonne connaissance du domaine et il est difficile de définir la notion de sémantique dans le cadre des commandes par rapport à si c'était une image ou du langage naturel.

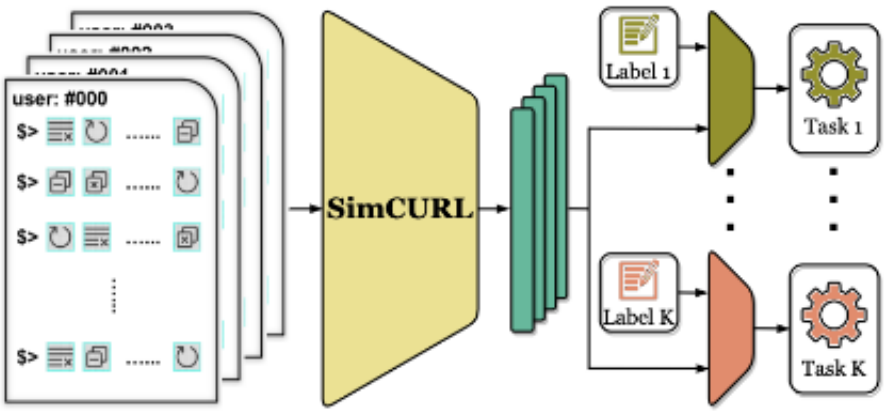


FIGURE 2.2 – Schéma présentant le système de l'article [2]

Il utilise dans leur approche le "self-supervised learning" (apprentissage auto supervisé) qui consiste à comparer à partir des données sans label à d'autres données avec label. Cela permet d'attribuer d'une certaine manière un label pour toutes les données se ressemblant. Il utilise toutefois comme dans la plupart des autres études, une suite de commande (avec la date d'utilisation) afin de pouvoir classifier les données qui seront utilisées pour la suggestion de commande.

2.2.2 Exemple : "Sequence Prediction Applied to BIM Log Data, an Approach to Develop a Command Recommender System for BIM Software Application"

L'article "Sequence Prediction Applied to BIM Log Data, an Approach to Develop a Command Recommender System for BIM Software Application" propose l'implémentation d'un outil de recommandation de commande pour les utilisateurs de logiciels de modélisation des données du bâtiments [quotation to add].

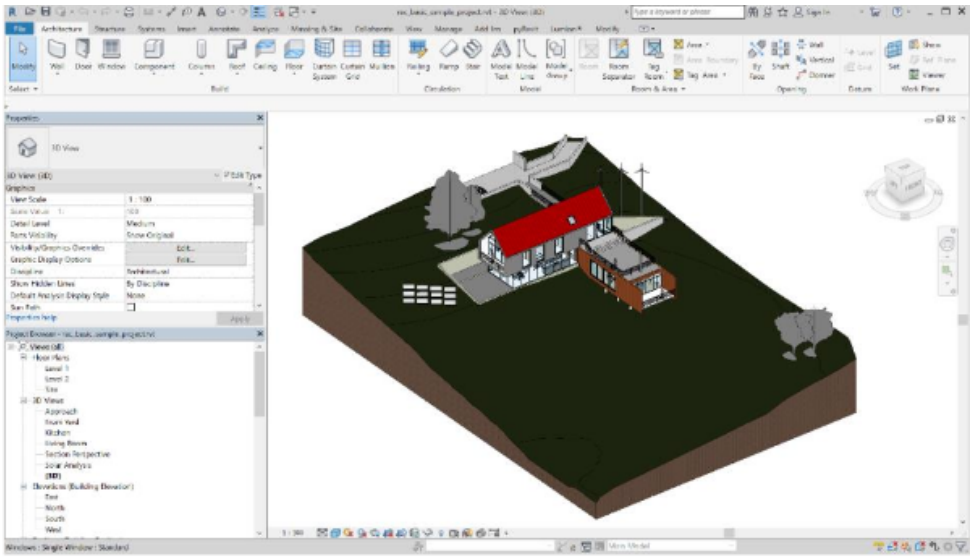


FIGURE 2.3 – AutoDesk Revit (image de l'article) [3]

L'approche consiste à utiliser les journaux ("logs") d'un logiciel tel que Autodesk Revit. Les données de ces journaux seront traitées puis employées pour développer un modèle de "Machine learning" pour suggérer des commandes à l'utilisateur afin d'améliorer leur rythme de travail. L'article emploie l'étude de l'apprentissage supervisé pour prédire une suite de commande qui serait utile à l'utilisateur. Notre cas serait de recommander la commande dans le cas où l'usager n'utiliserait pas la commande optimale ce qui est en partie similaire à l'étude proposer par l'article.

Cependant, la principale différence est que dans notre cas on utilise en entrée l'état du logiciel (image avant et après l'utilisation d'une commande) alors que l'article étudie l'historique des commandes.

Chapitre 3

Contribution

Dans cette partie nous allons présenter les différents points traités avec nos encadrants pour mener à bien notre projet. Il faut noter que le travail a été fait en collaboration avec des camarades du même Master : Nassim Ahmed-Ali et Christian Zhuang

3.1 Éditeur de texte

3.1.1 Les choix des commandes liés à l'éditeur de texte

Pour l'implémentation de l'éditeur de texte, nous avons décidé d'utiliser PyQt nous permettant surtout d'avoir un contrôle sur le logiciel crée. Le logiciel étant fait par nous même il est plus simple d'apporter des éléments extérieurs (modèle de classification et recommandeur). Nous avons 2 groupes de commandes choisi pour tester notre système.

1. Des commandes qui agissent directement sur le texte :

— Chercher et remplacer par la touche clavier "CTRL+R". Cette commande permet comme le nom l'indique de chercher un élément et de le remplacer. Il faudra noter que dans notre cas, notre version remplace toutes les occurrences pour permettre de mieux généraliser, car si le cas où pour toutes les occurrences cela fonctionne alors ce sera forcément le cas pour la commande élément par élément ; Par rapport au système, cette commande est une alternative plus rapide qu'à chercher chaque élément et les supprimer un à un par le mot que l'on souhaite écrire.

— Copier-Coller (CTRL+C et CTRL+V) des mots sélectionnés que l'ont souhaite ajouter à différents endroits. On pourrait généraliser cela pour les morceaux de texte mais il faudrait pouvoir utiliser des données fait par des utilisateurs car reconnaître ce qu'on sélectionne par un programme qui imite l'utilisateur aléatoirement n'est pas très efficace pour cette tâche ; Le système voudrait suggérer cette commande à la place d'écrire le mot à la main.

— Supprimer un mot (CTRL+Backspace). Le mot au niveau du curseur sera supprimé avec la commande. C'est une commande existante dans PyQt il faut donc juste pouvoir détecter lorsque l'utilisateur supprime un mot en appuyant plusieurs fois sur la touche "Backspace" afin de suggérer la commande.

2. Des commandes qui agissent sur le déplacement ou la sélection :

— Les déplacements mot par mot (CTRL+Left ou Right) qui remplace les déplacements caractère par caractère sur une certaine distance. Par exemple si l'utilisateur appuie plusieurs à droite, il faudra pouvoir lui recommander la commande de déplacement de mot. On a aussi inclut les déplacements vers début et fin de ligne qui nous semblait également intéressant pour accélerer le déplacement.

— Le même équivalent pour la sélection des mots. "CTRL+Shift+Left ou Right" pour les mots et "Shift+Home ou End" pour les lignes.

— Enfin, nous prenons également la commande de sélection du document entier avec "CTRL+A". Nous donnant ainsi de bonnes différences entre les commandes pour tester en partie la robustesse du système.

Les commandes de sélections sont déjà implémentés, mais il faut tout de même pouvoir détecter leur utilisation ainsi que les touches de claviers simples qui changerait le texte comme les lettres.

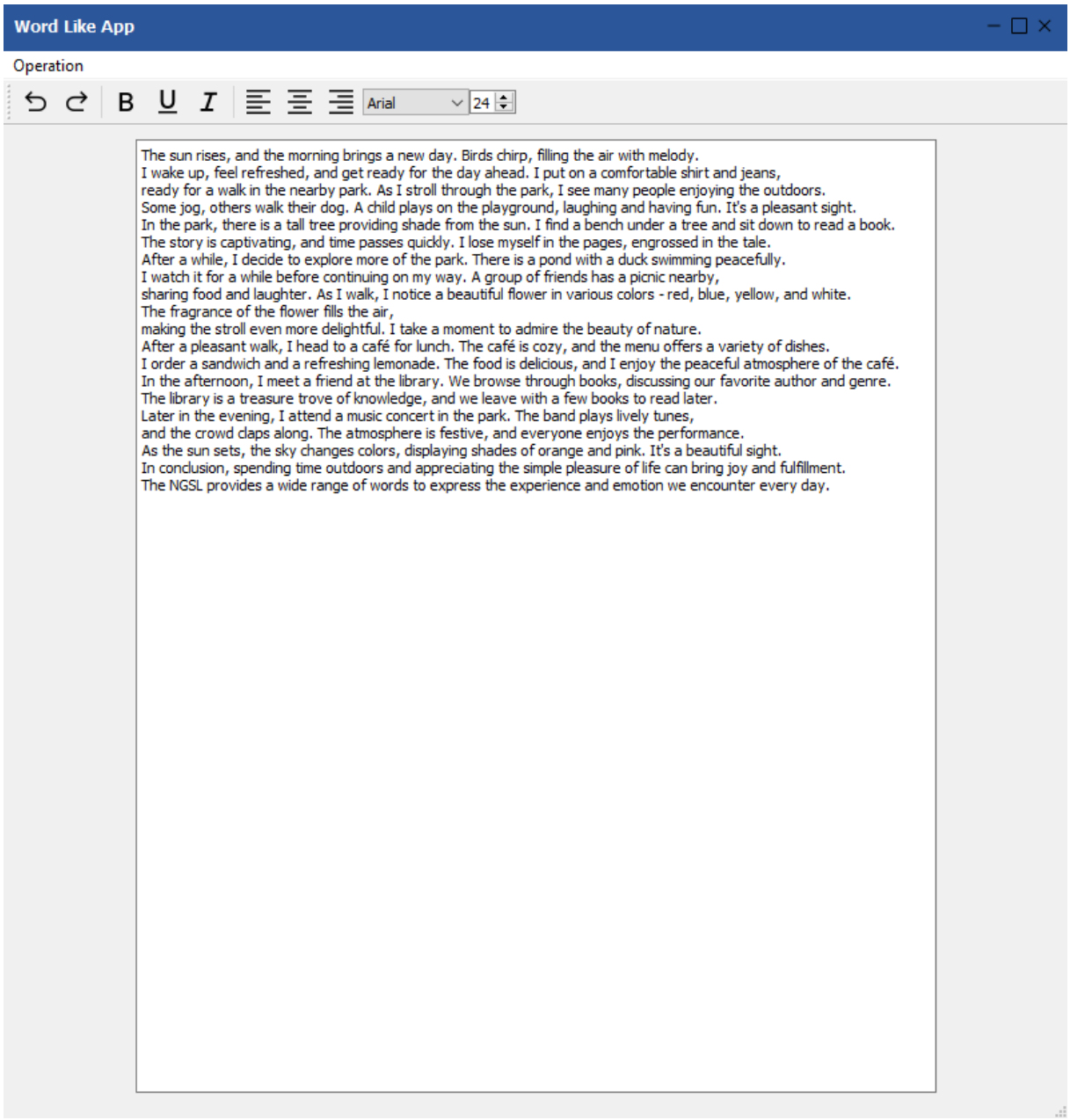


FIGURE 3.1 – Logiciel PyQt pour l'éditeur de texte

3.1.2 Gestion de la représentation des états et base de données avec player

L'étape la plus difficile pour la représentation d'un état pour le cas de l'éditeur de texte est un peu plus difficile que les deux autres qui utilisent directement des images. Contrairement aux autres, il faut définir la représentation. Les différentes représentations que nous avons testé pour cela sont :

— Représentation bag of word contaténée avec les positions de curseurs et de sélection. Le bag of word ("sac de mots") en lui même dépend du vocabulaire utilisé. Les informations du curseur sont la position dans le texte, la ligne et colonne ainsi que les positions de sélection (début vers fin).

— Deux vecteurs séparés avec en plus dans le vecteur bag of word, les mots non compris dans le vocabulaire et le nombre de caractères du document. C'est une seconde représentation des données vers laquelle nous nous sommes tourné, cela nous permet de tester la combinaison de plusieurs classifieurs et d'avoir potentiellement de meilleures performances. En séparant les deux vecteurs ont a comme vecteur de sélection : (position du curseur, ligne du curseur, colonne du curseur, début de sélection, fin de sélection) 2 fois, un avant la commande et un après avec un échantillon ci-dessous. Le vecteur bag of word étant illisible (dépendant d'un dictionnaire de mots), nous ne mettrons pas d'image mais des exemples peuvent êtres retrouvés sur le github mais l'idée est que pour chaque mot, son nombre d'occurrences sera présent dans le vecteur.

```
0,0,0,0,0,1000,10,11,1000,1000,deleteWord
1000,10,11,1000,1000,998,10,11,998,998,replace
998,10,11,998,998,994,10,7,994,994,deleteWord
994,10,7,994,994,987,10,0,987,987,deleteWord
987,10,0,987,987,986,9,42,986,986,deleteWord
986,9,42,986,986,986,9,42,986,986,replace
```

FIGURE 3.2 – Echantillon du vecteur sélection

A partir de ces représentations, il nous faut une base de données qui nous permettra d'apprendre dans le cas d'un classifieur de type classification supervisé. Pour mener à bien cela, nous avons fait un player qui doit imiter d'une façon l'utilisateur avec les différentes commandes qui sont étudiés pour le système.

Le player pour ce logiciel en particulier utilise directement PyQt avec la librairie QTest qui permet d'imiter les entrées claviers/souris, dans notre cas on se limitera aux entrées clavier pour simplifier la tâche et également car la souris sera en général plus rapide que le clavier dans la sélection si on laisse la souris. Le player pourra utiliser toutes les commandes de la partie 3.1.1. On a également ajouté une commande d'écriture pour permettre au modèle de reconnaître cela lorsque l'utilisateur écrit un mot qui n'existe pas dans le texte pour éviter de recommander des commandes qui n'ont pas de sens.

Pour son fonctionnement, le player choisira aléatoirement une commande parmis la liste et l'exécutera. Pour avoir des données assez différentes, on place le curseur à des positions de départ différentes choisi aléatoirement après qu'un certain nombre de commandes soit exécutées. Un point à noter ici est que dans la litérature, notamment ceux de l'état de

l'art, la génération des données est pris de journaux (logs) d'utilisateur comparé à notre cas où on génère automatiquement ces données sans l'utilisateur. Cependant, des données venant d'un utilisateur sont généralement mieux pour entraîner un classifieur face à un player qui utilise aléatoirement les commandes car ce sont des données qui apparaissent en pratique.

3.1.3 Modèles de classification

Les données générés avec les deux représentations choisies, nous avons testé différents modèles de classification. La suite étant très expérimentale, nous présentons plutôt les résultats liés aux modèles que nous avons étudiés. Cela nous permet de bien voir les limites de certaines représentations ou leurs points forts.

Les différents classifieurs utilisés sont les suivants :

— Un classifieur unique qui utilise la représentation qui concatène le bag of word avec le vecteur des positions et de sélection. C'est le cas de base vers lequel on se tourne car on aurait un classifieur supervisé général pour le logiciel.

— Un classifieur pour chacune des représentations vecteur bag of word et le vecteur des positions. Ce modèle est plutôt pour des vérifications de performances sachant que le bag of word et le vecteur de positions jouent des rôles différents selon le groupe de commandes.

— Enfin, le dernier modèle est la combinaison d'un modèle dit hardcodé selon des règles pour le vecteur de positions et d'un modèle appris pour le vecteur bag of word. Ce modèle découle plutôt du fait qu'il est simple de choisir des règles pour ce qui concerne la navigation et sélection dans un texte et qu'il est potentiellement mieux de faire cela pour éviter d'avoir de mauvaise recommandations. Par exemple, lorsqu'un utilisateur se déplacement suffisamment dans une direction, on peut suggérer en comparant les positions du curseur avant et après déplacement la commande de déplacement mot par mot.

Chaque classifieur entraîné est un réseau de neurones, ce choix est arbitraire mais est aussi du au fait que c'est également pour rester uniforme avec les modèles de réseau de convolution (CNN) utilisés par mes camarades. Dans le cas où on passerait des données de type images à la place, cela permettrait de facilement changer le modèle en ajoutant des couches de convolutions.

3.1.4 Choix du modèle final et difficulté des autres

Pour les différents modèles qui ont été appris (ceux qui ne sont pas hardcodés), suite à l'expérimentation, on obtient les résultats suivant sur les données de "Test/Validation" :

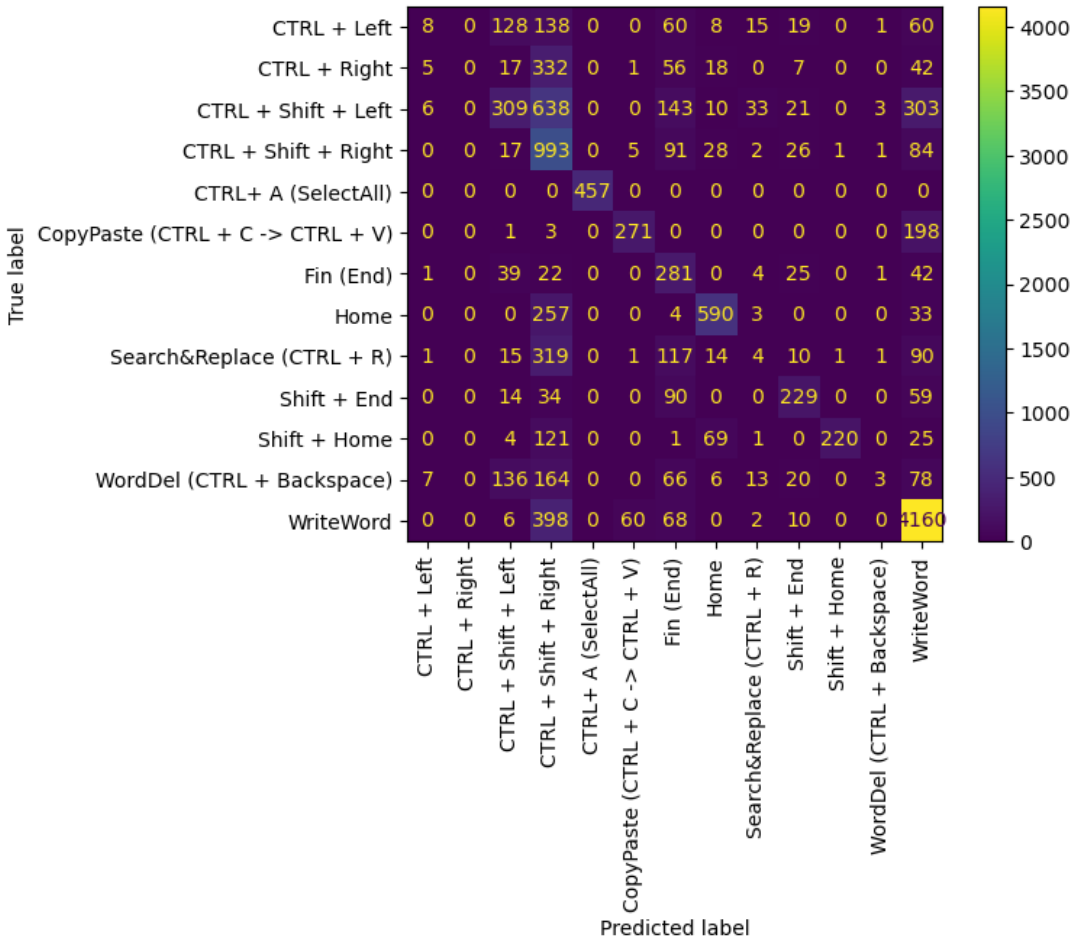


FIGURE 3.3 – Matrice de confusion pour le modèle unique de classifieur

	Précision	False Positive Rate
CTRL+Right (saut de mot droite)	28.6%	0.1%
CTRL+Left (saut de mot gauche)	0%	0%
CTRL+Shift+Left (selection de mot gauche)	45%	3.4%
CTRL+Shift+Right (selection de mot droite)	29%	21.5%
CTRL+A (Tout sélectionner)	100%	0%
CopyPaste (CTRL+C -> CTRL+V)	80.2%	0.5%
End (déplacement vers fin de ligne)	28.8%	5.7%
Home (déplacement vers début de ligne)	79.4%	1.3%
Search and Replace (CTRL + R)	5.2%	0.6%
Shift+End (Sélection jusqu'à la fin de la ligne)	62.4%	1.1%
Shift+Home (Sélection jusqu'au début de la ligne)	99.1%	0%
CTRL+Backspace (suppression d'un mot)	30%	0%
WriteWord (Ecriture d'un caractère)	80.4%	13%

Table 1 : Tableau des précision et taux de faux positif pour le modèle unique

La précision totale du modèle est de 60%, ce qui n'est pas très bon dans le cadre d'utilisation de notre système. On observe par le tableau qu'il y'a certains problèmes, le saut de mot vers la gauche n'est pas du tout reconnu et la sélection d'un mot vers la droite a un taux de faux positif très élevé. En observant la matrice de confusion liée au-dessus, la plupart des fausse prédictions se font par cette commande (il y a plus de données pour l'écriture de mots mais sachant que ce n'est utilisé que pour permettre au modèle de reconnaître cela, il n'y a pas vraiment de problème). Dans la globalité, on aurait très probablement du mal à faire fonctionner ce modèle en pratique

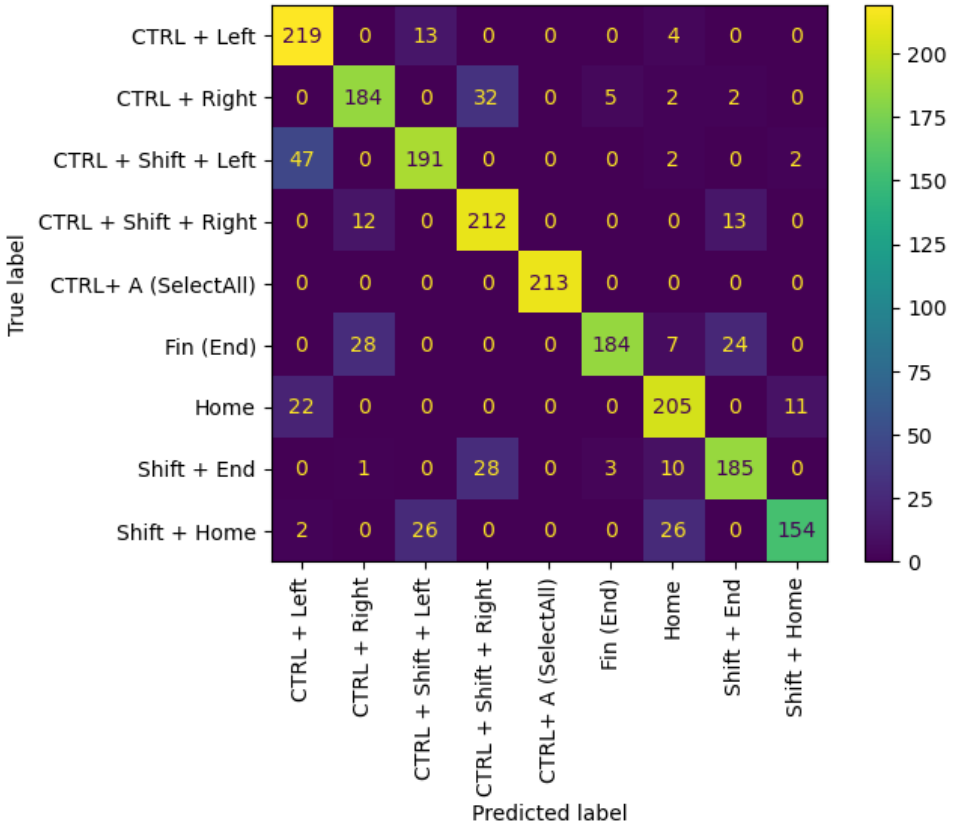


FIGURE 3.4 – Matrice de confusion pour le modèle sur les commandes de sélection

	Précision	False Positive Rate
CTRL+Right (saut de mot droite)	81.8%	2.2%
CTRL+Left (saut de mot gauche)	75.5%	3.8%
CTRL+Shift+Left (selection de mot gauche)	83%	2.1%
CTRL+Shift+Right (selection de mot droite)	77.9%	3.2%
CTRL+A (Tout sélectionner)	100%	0%
End (déplacement vers fin de ligne)	95.8%	0.4%
Home (déplacement vers début de ligne)	80.1%	2.7%
Shift+End (Sélection jusqu'à la fin de la ligne)	82.6%	2.1%
Shift+Home (Sélection jusqu'au début de la ligne)	92.2%	0.7%

Table 2 : Tableau des précision et taux de faux positif pour le modèle de machine learning sur le vecteur de sélection.

Le tableau pour la sélection donne des résultats très prometteur à première vue. La précision est plutôt élevé et le taux de faux positif est assez bas. De plus, la matrice de confusion montre de bons résultats globalement. En théorie, cela indique que ce serait potentiellement fonctionnelle pour la preuve de concept, mais en utilisant le modèle en pratique pour la sélection, on s'est rendu compte que cela ne fonctionne pas du tout. Par exemple, les déplacements sont confondues entre eux (droite et gauche). Une explication possible viendrait du fait que les données ont été générées aléatoirement et par conséquent, on a des données qui ne représentent pas les cas réels. Il nous faudrait des données générées grâce a des actions utilisateurs pour avoir des résultats plus concluant.

On notera que c'est surtout sur ce point qu'on se différencie des méthodes employées dans la littérature ; Ils utilisent des données d'historique utilisateur pour entraîner leur modèle alors qu'on essaye de générer ces données automatiquement qui est moins coûteux mais également difficile à utiliser. La précision globale est de 84.4% ce qui pour des données tests plutôt bons.

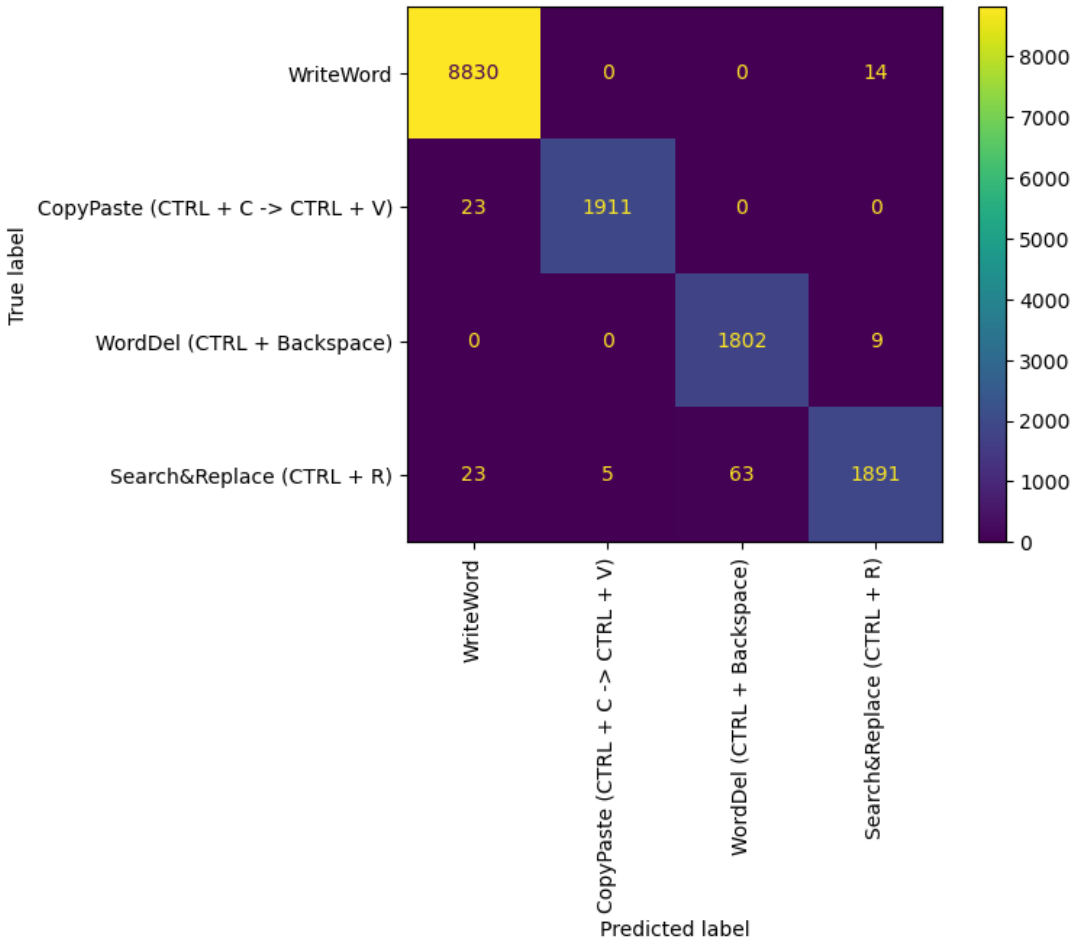


FIGURE 3.5 – Matrice de confusion pour le modèle sur les commandes du vecteur Bag of word

	Précision	False Positive Rate
CopyPaste (CTRL+C -> CTRL+V)	99.7%	0%
Search and Replace (CTRL + R)	98.8%	0.1%
CTRL+Backspace (suppression d'un mot)	96.6%	0.4%
WriteWord (Ecriture d'un caractère)	99.5%	0%

Table 3 : Tableau des précision et taux de faux positif pour le modèle de machine learning sur le vecteur bag of word.

Pour le vecteur bag of word, le modèle donne des résultats très élevé pour la précision et un taux de faux positif presque nul. En utilisant cela en pratique, les commandes agissant sur le texte directement donne bien ce à quoi on s'attend en majorité et ce résultat nous permet de déduire que ce modèle est bien utilisable pour la preuve de concept. On se retrouve donc à la fin avec seulement le vecteur bag of word utilisable et plutôt efficace en pratique avec une précision de 99.1% qui est le meilleur résultat obtenu par rapport au classifieur de sélection ou celui combinant les deux (sélection et bag of word) qui donnent tous les deux de mauvais résultats en pratique.

Pour le modèle codé avec des règles qui correspond à la partie du vecteur sélection, on n'a pas d'évaluation de type précision à faire, la décision donnée par ce classifieur dépend juste de la façon dont celui-ci est codé et donc de nos règles. L'évaluation qu'il serait possible de faire est de vérifier que nos règles correspondent aux bons critères (si cela correspond bien au comportement de la commande sur l'état).

Pour notre choix final, le modèle adopté est celui qui combine le classifieur appris sur le vecteur bag of word et un classifieur qui suit des règles codées en dur qui ne nécessite pas d'entraînement au préalable et donc pas de données. Cependant, le format du vecteur utilisé sera le même. Dans notre contexte, nous utiliserons toujours dans l'ordre le classifieur entraîné puis le classifieur codé en dur si rien n'est proposé par le classifieur de Machine Learning (supervisé) car le classifieur codé en dur garantissant une commande sans avoir un taux de confiance quelconque, on donne la priorité au classifieur supervisé qui ne sera pas interogé si le vecteur bag of word n'est pas modifié (aucune commande de traitement du texte n'a été appliquée).

3.2 Powerpoint, Photoshop et Vidéo

Nous allons présenter un peu plus généralement le travail réalisé dans les autres logiciel du système. Nous nous sommes repartis les tâches de tel sorte à ce que chaque personne traite un logiciel. Enfin, nous terminerons avec une briève présentation de la tâche finale de notre projet de stage qui est la vidéo faisant une démonstration du système.

3.2.1 PowerPoint

Cette partie du projet est réalisé en majorité par Christian Zhuang. La tâche ici est le même que pour la partie d'éditeur de texte mais pour le logiciel de présentation. Cela a été codé sous python également avec la librairie PyQt. Contrairement au vecteurs utilisés pour l'éditeur de texte, ce modèle utilise à la place directement une image du slide sur lequel l'utilisateur travail. Pour simplifier la génération des données, le player n'utilisera pas directement les touches du claviers ou la souris mais fait apparaître directement les

éléments sur le slide à partir des fonctions PyQt. Le classifieur utilisé est un réseau de convolution (CNN) car l'entrée était une image il est plus adéquat de faire cela. Pour ce logiciel nous avons finalement décidé d'utiliser un classifieur entièrement codé en dur par des règles car pas d'entraînement nécessaire et facilement codé.

3.2.2 Photoshop

Cette partie réalisée surtout par Nassim Ahmed-Ali est comparé aux autres un peu différent. En effet, le logiciel utilisée ici est directement photoshop. Nous avons décidé qu'il serait tout de même intéressant d'utiliser directement un programme existant pour au moins l'un des cas et c'est donc tombé sur celui-ci. Le modèle est très similaire à celui de powerpoint car on a comme entrée une image qui est la photo sur laquelle l'utilisateur modifie, on a un réseau de convolution (CNN) également ici. Cependant, la difficulté ici est qu'il n'est pas possible d'accéder aux commandes correspondant à l'état de la photo modifiée. Nous avons décidé de faire quelque chose qui permet de détecter certaines commandes en externe pour pouvoir réaliser cela malgré ces limites.

3.2.3 Vidéo

Pour finir, nous avons fait une vidéo faisait une démonstration du système DISCO (Contextually Improving Command Discoverability) pour la preuve de concept. La vidéo contient pour chaque logiciel 2 scénarios qui montre le fonctionnement du système de recommandation lorsque l'utilisateur utilise des commandes basiques de manières répétées alors qu'il existe une commande qui pourrait potentiellement rassembler cela. Cette vidéo marque la fin de notre travail actuel pour la preuve de concept.

Le lien vers la vidéo se trouve ICI

Chapitre 4

Conclusion

Pour conclure, nous avons donc conçu un système qui sert de preuve de concept pour un recommandeur de commandes contextuel. Pour cette partie concernant donc l'éditeur de texte, nous avons réussi à créer un player qui permet l'apprentissage pour quelques commandes qui agissent directement sur le texte avec une forte précision que ce soit à l'entraînement, test ou en pratique. Cependant, il semblerait comme vu précédemment que cela soit un peu difficile pour ce qui concerne la sélection, car la qualité des données ne reflètent pas assez bien la réalité et explique la difficulté d'un modèle entraîné en pratique.

Le système fonctionnant plutôt bien avec le classifieur entraîné sur le vecteur bag of word (commandes sur le texte) et le classifieur codé avec des règles (commandes de sélection), nous avons donc pu mener à bien ce travail pour la preuve de concept en créant la vidéo de démonstration.

Toutefois, il faut noter que même si contrairement aux autres systèmes de la littérature notre tâche en elle même ne joue pas sur des données générées par un utilisateur, avoir des données venant de comportements humain permettrait potentiellement de passer à un modèle entièrement entraîné et plus facile à coder dans la cas où il faut généraliser à toute les commandes d'un logiciel.

Par ailleurs, une futur perspective qu'on pourrait explorer dont on a surtout discuté avec nos encadrants Gilles Bailly et Julien Gori mais qui n'est pas abordé dans les exemples de ce projet : le code. Par exemple, l'utilisation de la bibliothèque pandas (avec des dataframes) où celui qui code utilise beaucoup de commandes basiques pour transformer un dataframe mais qu'on aurait pu directement utiliser une seule commande qui regrouperait les actions simples. Dans ce genre de système, on aurait sûrement besoin de méthodes de traitement du langage naturel (NLP) comme les transformer sous un modèle à la façon de ChatGPT.

Bibliographie

[1] Andy Cockburn - Carl Gutwin - Joey Scarr - Sylvain Malacria. « Supporting Novice to Expert Transitions in User Interfaces ». In : *ACM Comput. Surv.* (2014). URL : https://inria.hal.science/hal-02874746/document (pages 2, 3).

[2] Hang Chu - Amir Hosein Khasahmadi - Karl D.D. Willis - Fraser Anderson - Yaoli Mao - Linh Tran - Justin Matejka - Jo Vermeulen. « SimCURL : Simple Contrastive User Representation Learning from Command Sequences ». In : (2022). URL : https://arxiv.org/pdf/2207.14760.pdf (page 3).

[3] Radnia Ashkan. « Sequence Prediction Applied to BIM Log Data, an Approach to Develop a Command Recommender System for BIM Software Application ». In : *The University of North Carolina at Charlotte ProQuest Dissertations Publishing* (2021). URL : https://www.proquest.com/docview/2549243003?pq-origsite=gscholar&fromopenview=true (page 4).

Annexe A

Cahier des charges

A.1 Cahier des charges

A.1.1 Introduction

Ce projet a pour objectif de créer un assistant qui incite l'utilisateur à découvrir certaines commandes qui sont potentiellement meilleures que d'autres. Dans la suite du projet de Master I, certains points restent identiques : On aura un assistant qui analysera l'état du logiciel (image ou vecteur représentant cela) et proposera par un classifieur, une meilleure commande s'il en existe une (qui combinerait d'autres commandes basiques par exemple).

Pour mener à bien la preuve de concept, on implémentera cette assistant dans 3 logiciels différents : Un éditeur de texte (type word), Un programme de présentation (Google slide, Powerpoint) et enfin un logiciel de traitement d'images/dessins (photoshop). En prenant un exemple dans un éditeur de texte, l'utilisateur pourrait faire supprimer un mot en appuyant plusieurs fois sur "backspace" au lieu de juste appuyer sur "CTRL + backspace" qui supprime le mot directement et qui serait plus efficace au niveau du temps et effort.

Par rapport à d'autres travaux, il faut ici qu'on utilise l'état de l'application (le contexte en d'autres termes) pour comprendre l'intention de l'utilisateur plutôt que de suggérer une commande en fonction de ceux qui ont été utilisées par l'usager.

Le travail à réaliser sera représenté dans un premier temps par le schéma suivant (cela permettra de faire la base du projet) :

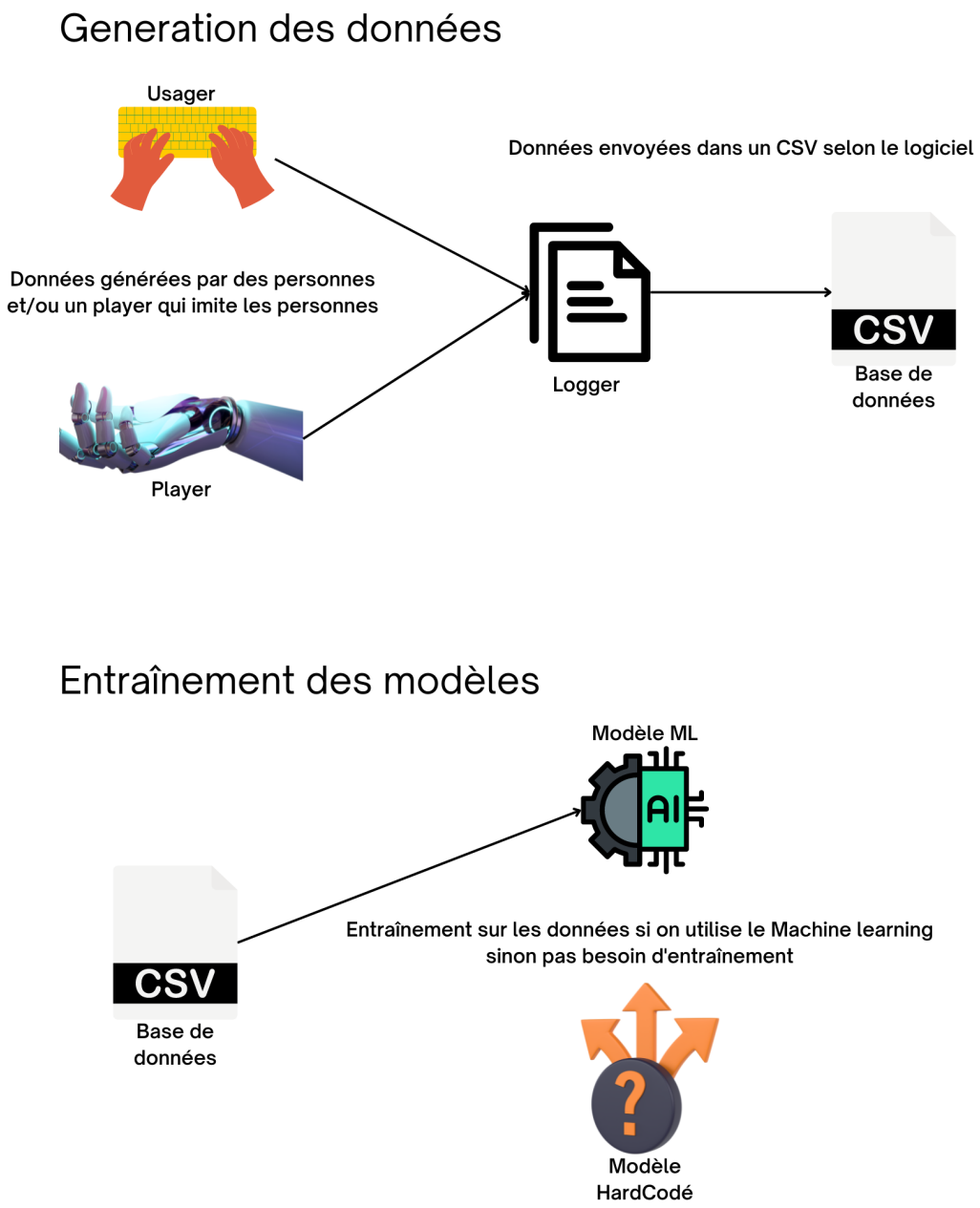


FIGURE A.1 – Partie offline : Génération des données et entaînement

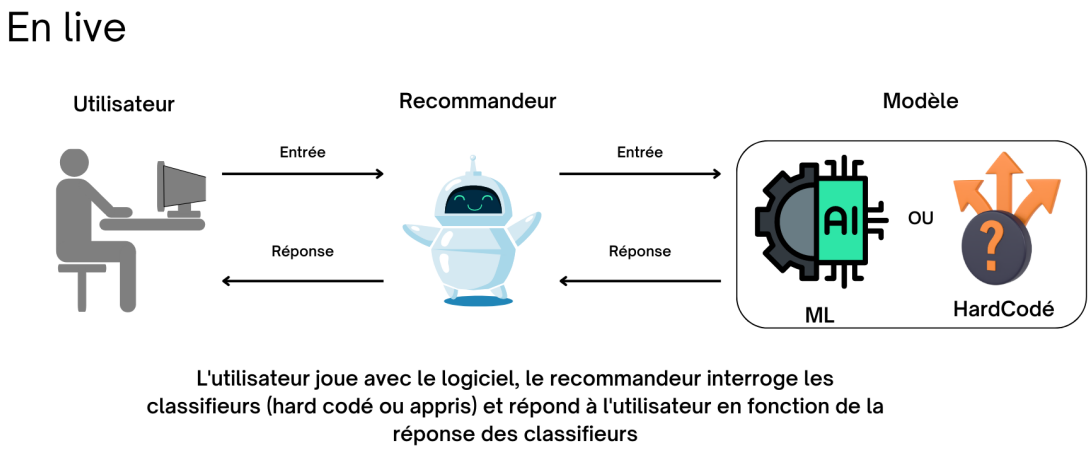


FIGURE A.2 – Utilisation du système en live

Le travail consiste à :

généraliser à différents contextes (Éditeurs de forme, éditeur photo, éditeur de texte).

- Réaliser un composant qui capture l'ensemble des commandes (pertinentes) de l'application
- Réaliser un modèle qui, à partir de deux états de l'application retourne la commande la plus appropriée.
- Réaliser un assistant qui repose sur les composants précédents pour présenter les raccourcis les plus pertinents aux bons moments.
- Réaliser des évaluations techniques pour comprendre les performances du modèle
- Réaliser des études utilisateurs pour évaluer la pertinence de l'approche.
- Réaliser des démonstrateurs pour étudier dans quelle mesure cette approche peut être

A.1.2 Les 3 Logiciels

Pour ce qui concerne les logiciels, l'éditeur de texte et le programme de présentation seront fait par nous même à l'aide de python (librairie PyQt) avec des fonctionnalités qui nous semblent pertinentes. On a grâce à cela, un meilleur contrôle sur ce qu'il passe dans les commandes. L'idée étant d'explorer les différentes commandes et de choisir ceux qui seraient les plus pertinents et reconnaissables par notre système.

Pour le logiciel de traitement d'images, nous avons décidé qu'il serait intéressant d'utiliser directement un des logiciels déjà entièrement fait entre krita (qui est open source) et photoshop. Le choix sera fait en fonction des fonctionnalités du logiciel.

Les décisions prisent sur quelles commandes nous semblaient intéressantes à utiliser sur l'assistant sont :

Pour le logiciel de type Word :

- Chercher et remplacer (CTRL + R dans notre logiciel)
- Les commandes de sauts entre les mots (CTRL + -> ou <-)
- Les commandes de sauts vers le début et vers la fin d'une ligne (home ou end sur un clavier qwerty)
- Les commandes de selections de mots (CTRL + SHIFT + <- ou ->) et du document (CTRL + A)
- La commande d'indentation (TAB)

Chacune de ces commandes peuvent être effectuées de façon un peu moins efficace (niveau temps et nombre de commandes)

Pour le Logiciel de type Powerpoint

- Commandes de groupage (CTRL + G) et dégroupage (CTRL + SHIFT + G) + Commandes alignements

Pour photoshop

- blur (gaussien, surface et iris)

Une grande partie du projet de stage a pour objectif d'implémenter les éléments de l'article suivant :

On devra également étudier différentes approches en ce qui concerne le modèle et ses données :

- Des modèles d'apprentissages et des modèles codés en dur.
- Les données seront générées par un outil qui imitera d'une façon l'utilisateur (utilisation aléatoire ou précise des commandes)
- Différents traitements sur les données ou même une façon particulière de les générer (cropping sur les images, normalisation, introduction de données fait à la main)

A partir de tous ces logiciels, le produit final qu'on doit fournir est une vidéo de scénarios pour chaque logiciel qui présente le concept de l'article (DISCO) donné par nos encadrants.

Au fur et à mesure du stage, d'autres points seront développés avec nos encadrants. Les tâches n'étant pas concises dû au fait qu'il faille expérimenter avec les différents logicielles et décider des commandes ainsi que des classifieurs les plus performants pour notre système.