

Projet 2A

HearMyHands

Programme de traduction de la Langue des Signes Française en temps réel



Dalyan Penisson
Arthur Pierre

Thibault Capdevielle
Nestor Coraboeuf
Nathan Dizy

Maxime D'Auria
Louna Pejot
Marius Demonfaucon

Killian Sevalle
François Delaunay

Titouan Desailly
Kaelig Leray

Tutrice pédagogique : Hélène Pérennou – Promotion Projet PeiP 2A – 2025-2026

Table des matières

1	Résumé	4
2	Aspects techniques et scientifiques	6
2.1	Contexte, problématique et plan	6
2.2	État de l'art et approche théorique	8
2.3	Architecture générale de la solution	10
2.4	Module de vision	13
2.5	Module de traduction	17
2.6	Hypothèses retenues	26
2.7	Constitution et usage du dataset	27
2.8	Mise en œuvre expérimentale	27
2.9	Résultats et analyse	28
2.10	Perspectives	30
2.11	Conclusion générale	33
3	Travaux complémentaires	34
3.1	Essais réalisés en parallèle	34
3.2	Recherche d'hyperparamètres du MLP	34
3.3	Choix de la loss du Modèle 1	35
3.4	Segmentation vidéo et annotations COCO	35
3.5	Exploration Blender	37
4	Aspects gestion de projet	39
4.1	Organisation de l'équipe	39
4.2	Outils utilisés	40
4.3	Difficultés rencontrées	40
4.4	Analyse des risques	41
4.5	Évolution du planning	42
4.6	Bilan de gestion	43
	Remerciements	44
5	Bibliographie	45
6	Glossaire	48

A	Annexe : exemple de structure JSON d'annotation	51
B	Annexe : questionnaire de validation du besoin	53
C	Annexe : dépôts du projet	54

1

Résumé

En français.

HearMyHands est notre projet d'intelligence artificielle visant à traduire la Langue des Signes Française (LSF) en texte en temps réel à partir d'une simple webcam. Nous répondons à une problématique d'inclusion : la communication entre personnes sourdes ou malentendantes et personnes entendant reste souvent freinée par le manque d'interprètes disponibles et par le faible nombre de personnes maîtrisant la LSF [1]. Notre démarche s'inscrit principalement dans l'Objectif de Développement Durable n°10 de l'ONU, qui vise à réduire les inégalités, et ouvre également une perspective éducative liée à l'ODD 4 [2, 3].

Nous développons une solution reposant sur une architecture en deux modules et sur une interface web. Le navigateur capture le flux de la webcam, l'envoie à un serveur, puis affiche en retour le squelette détecté et le texte reconnu. Le premier module, dit module de vision, extrait d'un flux vidéo les points caractéristiques du corps, des bras et des mains afin de produire une représentation simplifiée sous forme de squelette. Le second module, dit module de traduction, analyse l'évolution de ce squelette pour reconnaître les lettres de l'alphabet en LSF, y compris les lettres dynamiques. Cette séparation réduit la complexité du problème, car nous n'apprenons pas directement une correspondance entre une vidéo brute et du texte : nous travaillons d'abord sur des coordonnées plus compactes, plus faciles à traiter et plus adaptées à nos ressources de calcul.

Nous avons construit HearMyHands autour d'un périmètre maîtrisé : la reconnaissance de l'alphabet en LSF, avec des lettres statiques et des lettres dynamiques. Dans le cadre de ce rapport, nous ne cherchons donc pas à reconnaître des mots, des phrases ou des échanges complets. Ce choix nous permet de traiter à la fois des configurations de main fixes et des gestes en mouvement, tout en conservant une architecture compréhensible, testable et extensible. Le rapport présente également notre démarche de projet : validation du besoin, adaptation du périmètre, organisation d'une équipe nombreuse, gestion des risques et réflexion sur un modèle économique responsable.

In English.

HearMyHands is our artificial intelligence project aiming to translate French Sign Language (LSF) into text in real time using a simple webcam. We address an inclusion issue : communication between deaf or hard-of-hearing people and hearing people is still often limited by the lack of available interpreters and by

the small number of people who master *LSF* [1]. Our approach mainly contributes to the United Nations Sustainable Development Goal 10, reduced inequalities, and also opens an educational perspective linked to SDG 4 [2, 3].

We develop a solution based on a two-module architecture and a web interface. The browser captures the webcam stream, sends it to a server, then displays the detected *skeleton* and the recognized text in return. The first module, called the vision module, extracts body, arm and hand *keypoints* from the video stream in order to produce a simplified *skeleton* representation. The second module, called the translation module, analyses the evolution of this *skeleton* to recognize letters from the *LSF* alphabet, including dynamic letters. This separation reduces the complexity of the problem, because we do not directly learn a mapping between raw video and text : we first work on more compact coordinates, which are easier to process and better suited to our computing resources.

We built *HearMyHands* around a controlled scope : recognition of the *LSF* alphabet, with both static and dynamic letters. In this report, we therefore do not aim to recognize words, sentences or full conversations. This choice allows us to handle both fixed hand configurations and moving gestures, while keeping an architecture that is understandable, testable and extensible. The report also presents our project approach : need validation, scope adaptation, organization of a large team, risk management and reflection on a responsible economic model.

2

Aspects techniques et scientifiques

2.1. Contexte, problématique et plan.

2.1.1. Contexte social et besoin.

La Langue des Signes Française est une langue visuelle et gestuelle utilisée par une partie des personnes sourdes et malentendantes. Elle ne se limite pas à une transcription gestuelle du français : elle possède sa propre grammaire, sa propre syntaxe et mobilise simultanément les mains, le corps, le regard et les expressions du visage. Les prises de position récentes de la Fédération Nationale des Sourds de France rappellent l'importance de distinguer interprétation, traduction et accessibilité réelle pour éviter une approche réductrice de la LSF [4]. La richesse des émotions, des mouvements du corps et de l'espace de signation doit donc être correctement traduite pour préserver le sens du message, ce qui rend l'automatisation à la fois difficile et socialement importante.

Dans la vie quotidienne, beaucoup d'échanges simples deviennent complexes lorsqu'aucun interlocuteur ne maîtrise la LSF : accueil administratif, rendez-vous médical, scolarité, commerce, réunion de travail ou échange informel. Cette difficulté rejoint les principes de la Convention internationale relative aux droits des personnes handicapées, qui insiste sur l'accès à la communication, à l'information et à la participation sociale [5]. Les interprètes professionnels jouent un rôle essentiel, mais leur disponibilité reste limitée et leur intervention n'est pas toujours possible pour des interactions courtes ou spontanées ; la FNSF souligne notamment les tensions autour de l'interprétation, de la traduction et de l'accessibilité dans plusieurs situations de vie [4]. HearMyHands se place dans cet espace : proposer un outil d'assistance accessible, utilisable avec un ordinateur standard et une webcam.

Le besoin concerne aussi le cadre éducatif. Les difficultés d'accès à la LSF et aux adaptations nécessaires peuvent limiter l'autonomie des élèves et étudiants sourds ou malentendants, ce que la FNSF documente dans sa prise de position sur l'éducation [6]. Cette dimension explique pourquoi HearMyHands ne vise pas seulement une démonstration technique : le projet pose aussi les bases d'un outil qui soutient l'apprentissage et la sensibilisation du public entendant.

Pour éviter de construire le projet uniquement sur une intuition, nous avons mené une première étude de terrain. Elle a pris la forme d'un questionnaire ciblé et de prises de contact avec des associations spécialisées dans la surdité et la LSF. Les questions diffusées sont reprises en annexe B. Malgré plusieurs mails envoyés à des associations, nous n'avons pas obtenu de retour direct de leur part dans le délai du projet. Les réponses exploitables proviennent donc de quatre contributions : deux chaînes YouTube

spécialisées dans la langue des signes, Coralie Castel et Je m'IniSigne, ainsi que des réponses issues de groupes Facebook et forums spécialisés comme *Sourds Oralistes*, *Forum AFIDEO*, *les sourds a la France*, *France sourds*, *Les SOURDS de la DIVERSITÉ*, *Tous Ensemble !* et *Sourds et Malentendants* [7, 8, 9, 10, 11]. Même si cet échantillon reste limité, ces retours ont confirmé l'intérêt d'une solution non intrusive. Ils ont aussi rappelé que les mains ne suffisent pas à porter tout le sens : expressions faciales, posture, mouvement et espace de signation jouent un rôle majeur.

Les réponses au questionnaire mettent en évidence plusieurs points structurants. Les interprètes humains restent perçus comme le moyen de traduction le plus efficace, tandis que les outils automatiques sont encore jugés peu répandus ou limités. L'erreur la plus souvent relevée est la traduction mot à mot, car elle ignore la syntaxe propre à la LSF, les expressions du visage, les émotions et le contexte. Les personnes interrogées insistent aussi sur l'importance du haut du corps et sur le rôle central du mouvement. Elles rappellent enfin qu'il existe des variantes régionales, générationnelles et informelles, et qu'un même signe peut changer de sens selon la situation.

Cette phase de validation du besoin a aussi joué un rôle méthodologique. Elle nous a obligés à cadrer notre démonstrateur sans réduire l'ambition sociale du projet.

2.1.2. Objectifs de développement durable.

Le projet s'inscrit principalement dans l'ODD 10, *Inégalités réduites*, car il vise à réduire une barrière de communication qui peut limiter l'accès aux services, à l'éducation et à l'emploi [2]. Il rejoint aussi l'ODD 4, *Éducation de qualité*, car le même socle technique sert à proposer un outil d'apprentissage interactif de la LSF [3]. L'application peut ainsi accompagner un geste, proposer des exercices progressifs et aider le grand public à découvrir la langue.

L'intérêt du projet est donc double. D'une part, HearMyHands cherche à faciliter des échanges ponctuels dans des situations concrètes où la présence d'un interprète n'est pas réaliste. D'autre part, le projet peut contribuer à rendre la LSF plus visible auprès d'un public entendant qui la connaît souvent mal. Cette visibilité compte, car l'inclusion ne dépend pas uniquement d'un outil de traduction, mais aussi d'une meilleure compréhension des besoins, des usages et de la culture des personnes sourdes ou malentendantes.

Nous avons également veillé à ce que la solution envisagée ne renforce pas une inégalité matérielle. L'objectif est d'utiliser un matériel déjà largement disponible pour proposer une aide simple, rapide et peu coûteuse.

2.1.3. Problématique.

La problématique retenue est la suivante :

Comment traduire en temps réel la Langue des Signes Française en texte avec une simple webcam, une interface accessible et une puissance de calcul déportée côté serveur ?

Cette formulation impose plusieurs contraintes. Le système doit traiter une information visuelle fortement variable et rester utilisable depuis un ordinateur ou un téléphone. La puissance de calcul n'a pas vocation à dépendre du terminal de l'utilisateur : elle peut être portée par un serveur, aujourd'hui déployé sur [Google Cloud](https://hearmyhands.asia/) et accessible à l'adresse <https://hearmyhands.asia/>. La problématique devient donc autant une question d'architecture que de modèle.

Elle impose aussi de définir un périmètre réaliste. Le cadrage retenu permet de valider l'approche sur un alphabet complet, avec des lettres statiques et des lettres dynamiques, avant d'envisager des usages linguistiques plus larges. Le projet a donc été construit par notre équipe de douze membres, avec une montée en compétence progressive en Python, vision par ordinateur et [apprentissage profond](#), et avec

l'appui ponctuel de personnes ressources du LS2N. La réponse à cette problématique n'est pas qu'un choix d'algorithme ; elle relève aussi d'une réflexion d'ingénierie sur les données, le temps réel, le déploiement serveur et la cohérence des modules.

2.1.4. Plan de réponse.

La suite du rapport présente d'abord les solutions existantes et les notions théoriques nécessaires. Elle détaille ensuite l'architecture retenue, les hypothèses de développement, le protocole de constitution du [dataset](#), les choix de modèles et la chaîne de traitement. Le rapport analyse enfin les limites, les perspectives d'évolution, le modèle économique envisagé et la gestion de projet.

2.2. État de l'art et approche théorique.

2.2.1. Solutions existantes.

Plusieurs familles de solutions permettent d'aborder la reconnaissance de la langue des signes. La première repose sur des capteurs portés par l'utilisateur, par exemple des gants connectés mesurant la flexion des doigts, l'orientation de la main ou le mouvement. Le projet SignAloud, récompensé par le prix étudiant Lemelson-MIT, illustre cette approche : chaque gant contient des capteurs, transmet les données par Bluetooth et associe certains gestes à du texte ou à de la parole [12]. Ces dispositifs montrent qu'une mesure instrumentée du mouvement peut simplifier la reconnaissance, mais ils imposent un équipement spécifique, visible et parfois coûteux. Ils captent surtout les mains, alors qu'une langue des signes mobilise aussi le corps, le regard, l'espace et les expressions non manuelles. Cette limite explique pourquoi nous n'avons pas retenu une solution par gants.

La seconde famille repose sur la vision par ordinateur. Une caméra observe l'utilisateur, puis un algorithme extrait les informations utiles : position du corps, orientation des bras, localisation des mains et configuration des doigts. Les travaux récents utilisent soit des images brutes, soit des [points clés](#) extraits par des bibliothèques comme [MediaPipe](#), OpenPose ou des modèles de pose dédiés. Cette seconde famille est plus exigeante en traitement d'image, car elle dépend du cadrage, de la lumière et des occultations, mais elle présente un avantage décisif : elle fonctionne avec une webcam standard. Elle correspond donc mieux à notre objectif d'accessibilité.

Les [datasets](#) existants montrent l'écart entre reconnaissance de signes isolés et traduction de phrases complètes. WLASL propose un grand corpus de signes en [ASL](#) isolés au niveau du mot, utile pour comparer des modèles de reconnaissance lexicale [13]. How2Sign va plus loin avec plus de 80 heures de vidéos [ASL](#) continues, accompagnées de transcriptions et de modalités complémentaires [14]. Pour l'allemand signé, RWTH-PHOENIX-Weather 2014T fournit un corpus vidéo, glosses et traductions, devenu un benchmark majeur pour la traduction continue [15, 16]. Pour la [LSF](#), des ressources récentes comme Mediapi-[RGB](#) et Matignon-[LSF](#) montrent que la constitution de corpus vidéo-texte français progresse, avec des dizaines d'heures de vidéos annotées ou sous-titrées [17, 18].

Enfin, des systèmes de traduction plus ambitieux cherchent à apprendre directement une correspondance entre vidéo et texte. Les Sign Language Transformers de Camgoz et al. constituent une référence importante : ils combinent reconnaissance de glosses et traduction dans une architecture Transformer, et montrent que l'information intermédiaire de type gloss améliore fortement la traduction par rapport à un passage direct vidéo-vers-texte [19]. Ce point est central pour notre projet. Une approche directe vidéo-vers-texte aurait demandé un corpus massif, aligné, annoté et représentatif de la [LSF](#), ainsi qu'une puissance de calcul et une expertise très supérieures à notre cadre de projet. Nous avons donc retenu une architecture intermédiaire : transformer la vidéo en coordonnées structurées, puis appliquer des modèles plus légers sur ces coordonnées. Cette décomposition n'est pas un renoncement, mais un choix d'ingénierie cohérent avec l'état de l'art et avec nos ressources.

2.2.2. Représentation par squelette.

Une image brute contient de nombreuses informations inutiles pour notre objectif : arrière-plan, texture des vêtements, variations d'éclairage ou détails sans lien avec le geste. Nous réduisons donc l'information visuelle à un **squelette**, c'est-à-dire à un ensemble de **points clés**. Pour le haut du corps, ces points décrivent le cou, les épaules, les coudes, les poignets et les hanches. Pour les mains, ils décrivent les 21 articulations détectées par **MediaPipe**.

Cette représentation compacte présente trois avantages. Elle réduit fortement la dimension des données : une image couleur de 480×360 pixels contient plus de 500 000 valeurs numériques, alors qu'une main représentée par 21 points en deux dimensions ne contient que 42 coordonnées. Le **squelette** facilite aussi l'affichage d'un retour visuel dans l'interface et permet d'entraîner ou d'utiliser des modèles plus légers. Il introduit en revanche une dépendance forte à la qualité de détection ; si un poignet ou une main est mal localisé, l'erreur se propage vers la reconnaissance de lettre.

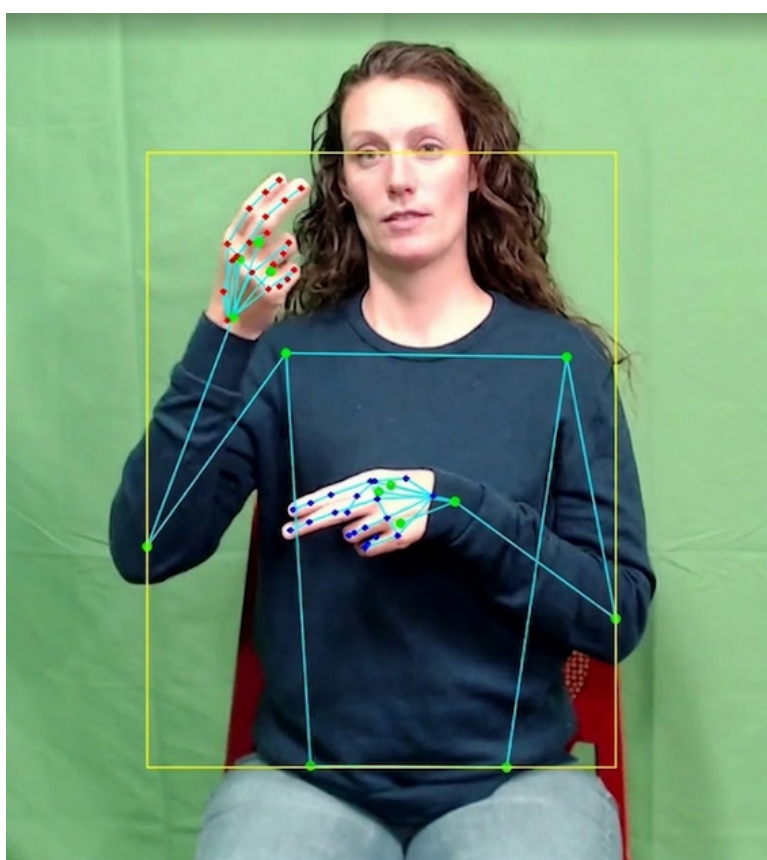


Figure 2.1 – Exemple de représentation par squelette obtenue sur une image issue de How2Sign : les points du haut du corps et des mains sont extraits par notre chaîne combinant MediaPipe et la partie custom du Modèle 1.

2.2.3. Choix des modèles.

Notre solution combine deux niveaux de traitement. Le premier niveau, appelé **Modèle 1**, correspond au module de vision : il extrait la posture du haut du corps avec notre modèle de détection de points clés, puis complète cette représentation avec **MediaPipe Hands** [20, 21] pour obtenir les articulations des mains. Le second niveau, appelé **Modèle 2**, correspond au module de traduction : il exploite ces coordonnées avec un **perceptron multicouche** pour reconnaître les lettres statiques et avec une branche

temporelle fondée sur un **GRU** pour les lettres dynamiques. Un perceptron multicouche est un réseau de neurones simple où les valeurs d'entrée traversent plusieurs couches de calcul avant de produire une classe de sortie. Un GRU, ou *Gated Recurrent Unit*, est un réseau récurrent conçu pour traiter une séquence dans le temps : il conserve une mémoire interne des frames précédentes afin de reconnaître un mouvement plutôt qu'une seule posture.

Le choix d'un **MLP** pour les lettres statiques est adapté au périmètre traité. Il reçoit des coordonnées de main normalisées et classe une posture isolée. Nous nous sommes aussi appuyés sur des supports pédagogiques pour stabiliser le vocabulaire, comprendre les bases mathématiques et clarifier les notions autour des perceptrons multicouches [22, 23]. Même lorsque les 26 lettres sont présentes dans les classes, celles qui dépendent fortement d'un mouvement, comme J ou Z, restent difficiles à reconnaître avec une seule posture ; leur traitement justifie donc la branche temporelle du Modèle 2.



Figure 2.2 – Principe général de HearMyHands : acquisition par webcam, extraction des points clés, traduction par le Modèle 2 puis affichage du texte reconnu.

2.2.4. Justification de la décomposition.

Nous ne cherchons pas à faire apprendre au modèle toute la vidéo brute. Une telle stratégie forcerait le réseau à comprendre simultanément la personne, le fond, la lumière, la posture, les mains et le sens linguistique. En décomposant le problème, nous séparons les difficultés : le module de vision extrait les **points caractéristiques**, puis le module de traduction interprète une représentation numérique beaucoup plus simple. Cette séparation rend la solution plus lisible, plus contrôlable et plus compatible avec un fonctionnement en temps réel.

2.3. Architecture générale de la solution.

2.3.1. Vue d'ensemble.

Les éléments de code cités dans cette partie renvoient au dépôt public HearMyHands, disponible à l'adresse github.com/nmqx/hearmyhands. Le dépôt complet est également référencé en annexe C avec les autres livrables du projet.

HearMyHands est organisé selon une architecture **client-serveur**. L'application web **Flask**, située dans le dossier `hearmyhands/`, sert les pages **HTML**, gère l'interface utilisateur et reçoit les images de la webcam par **Socket.IO**. En développement local, elle peut être lancée directement avec `python hearmyhands/app.py`. En déploiement, **Flask** reste le framework applicatif, tandis que **Gunicorn** avec un worker compatible **WebSocket** sert l'application en production. Dans la version actuelle, l'**inférence**, c'est-à-dire l'utilisation d'un modèle déjà entraîné pour produire une prédiction sur une nouvelle image, est chargée directement dans ce même processus grâce à `HmH/inference.py`. Cette organisation évite un aller-retour **HTTP** interne et améliore la réactivité. Le mode séparé reste possible avec `HmH/api.py`, qui ex-

pose les routes historiques `hearmyhands.asia/model_predict` et `hearmyhands.asia/sign_predict` lorsque le modèle doit tourner sur une autre machine.

Le navigateur envoie des images **JPEG** compressées en binaire. Le frontend redimensionne chaque frame à une largeur de 480 pixels avec une qualité **JPEG** de 0,7. Cette valeur n'est pas une précision du modèle, mais un paramètre de compression : plus elle est élevée, plus l'image est fidèle, mais plus elle est lourde à transmettre. En conditions réelles, le système fonctionne plutôt autour de 10 à 15 **images par seconde**, selon le serveur, le réseau et la charge des modèles. Cette cadence constitue un compromis acceptable pour la démonstration actuelle : elle limite la bande passante tout en conservant assez d'information temporelle pour tester les lettres dynamiques.

Le pipeline logiciel se résume ainsi :

1. capture du flux vidéo par le navigateur ;
2. compression et envoi de la frame par **Socket.IO** ;
3. traitement de la frame par le moteur d'**inférence**, directement dans l'application ou via l'**API** modèle en mode séparé ;
4. extraction des points du haut du corps par la partie custom du **Modèle 1** ;
5. détection des mains par la partie **MediaPipe** du **Modèle 1** ;
6. selon le mode sélectionné, classification de la première main détectée avec le **MLP** du **Modèle 2** ou constitution d'une séquence temporelle de **keypoints** pour le **GRU** Ocarina ;
7. retour des **keypoints**, lettre prédite et scores de confiance vers l'interface ;
8. affichage du **squelette**, des mains et de la suite de lettres reconnues.

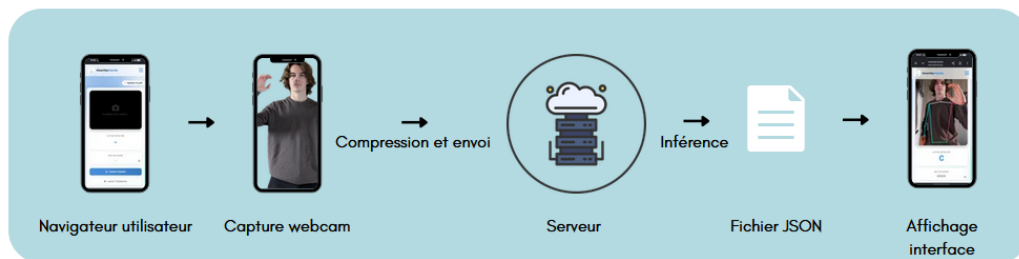


Figure 2.3 – Pipeline d'utilisation : le navigateur capture la webcam, transmet les frames au serveur, reçoit les résultats d'inférence et met à jour l'interface.

2.3.2. Interface web.

L'interface web comporte trois pages principales : une page d'accueil, une page de traduction et une page d'apprentissage. La page de traduction est la partie fonctionnelle centrale. Elle active la webcam, affiche le flux vidéo, superpose un canvas pour dessiner les points détectés et affiche la lettre courante ainsi que la suite de lettres reconnues. Un bouton permet de basculer entre le mode statique, fondé sur le **MLP**, et le mode dynamique, fondé sur le **GRU** Ocarina.

La page d'apprentissage reprend le même principe d'analyse par webcam, mais dans un usage guidé. Elle propose des cartes de révision pour l'alphabet : l'utilisateur observe une vidéo modèle, reproduit la lettre, puis reçoit un retour à partir de la prédiction du système. Elle comporte aussi un quiz avec trois modes d'entraînement. L'utilisateur doit reconnaître des signes affichés ou effectués, puis répondre à l'aide d'un plateau de choix, ce qui permet de travailler à la fois la production et la reconnaissance des

lettres. Cette partie transforme le démonstrateur en support d'entraînement progressif, en plus de la traduction.

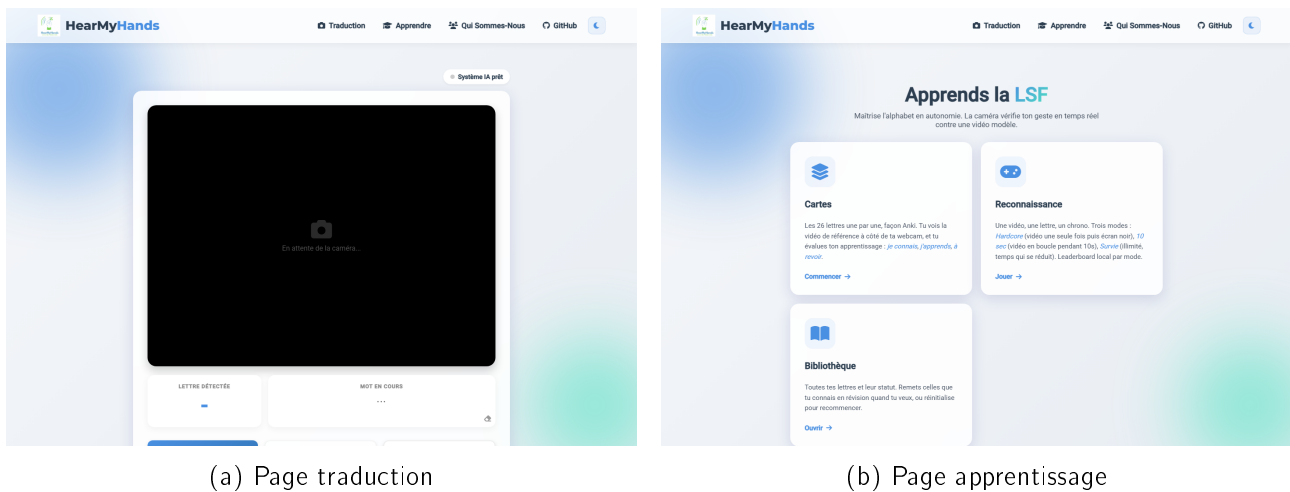


Figure 2.4 – Captures de l'interface web publique : la page de traduction affiche le flux caméra et les sorties du modèle, tandis que la page d'apprentissage regroupe les cartes de révision, les exercices de reconnaissance et la bibliothèque.

Le script `JavaScript` applique un mécanisme de stabilisation avant d'ajouter une lettre à l'historique reconnu. En mode statique, la prédiction doit dépasser un seuil de confiance élevé, typiquement 0,75 ou 0,80 selon les réglages de démonstration. Elle est ensuite placée dans une mémoire courte de cinq images : une lettre n'est validée que si elle apparaît au moins trois fois dans cette fenêtre. Lorsque le modèle hésite fortement entre deux lettres, l'affichage peut signaler cette incertitude au lieu de présenter la prédiction comme parfaitement fiable. En mode dynamique, l'envoi des frames est d'abord contrôlé par un seuil visuel : la main doit passer au-dessus d'une ligne affichée sur la vidéo pour armer la capture. Cette logique évite d'alimenter le modèle temporel avec des frames inutiles lorsque l'utilisateur n'est pas en train de signer. Le résultat du `GRU` est ensuite filtré avec un seuil de confiance de 0,7 et un délai anti-doublon pour éviter de répéter plusieurs fois le même signe temporel.

2.3.3. API modèle.

Le moteur d'inférence centralise le chargement et l'exécution des modèles. En mode intégré, il est importé directement par l'application web. En mode séparé, l'API modèle propose une adresse de traitement, appelée route `hearmyhands.asia/model_predict`. Concrètement, l'application web envoie une image à cette adresse, puis le serveur renvoie une réponse structurée. L'image reçue est décodée avec OpenCV avant les prétraitements nécessaires au modèle. La réponse `JSON` contient notamment `keypoints` pour le haut du corps, `hands` pour les `keypoints` des mains, `letter` pour la lettre reconnue, `confidence` pour le score associé, ainsi que les dimensions de l'image utilisées pour normaliser les séquences temporelles.

La route `hearmyhands.asia/sign_predict` traite la branche temporelle. Elle reçoit une séquence de frames déjà transformées en coordonnées de main prétraitées, cohérentes avec l'entraînement. Cette route renvoie la classe prédite par Ocarina et sa confiance. Lorsque le modèle `GRU` n'est pas chargé, la route renvoie une erreur contrôlée et l'interface peut continuer à fonctionner en mode statique.

Une route `hearmyhands.asia/healthz` permet aussi de vérifier l'état du système : chargement de la partie custom du `Modèle 1`, disponibilité de `MediaPipe`, disponibilité du classifieur de lettres statiques, disponibilité du classifieur temporel et appareil utilisé pour l'inférence. Une page `hearmyhands.asia/monitor`,

appuyée par la route `hearmyhands.asia/stats`, facilite le suivi de la charge serveur pendant les démonstrations.

2.4. Module de vision.

2.4.1. Dataset et prétraitement de la partie custom du Modèle 1.

La partie custom du [Modèle 1](#) a pour objectif de suivre 9 points d'intérêt du haut du corps à partir d'un flux vidéo. Pour construire les premières annotations, nous avons utilisé une stratégie de [pseudo-labélisation](#) : [BlazePose Heavy](#), fourni par [MediaPipe](#), extrait 33 repères corporels sur des vidéos tournées sur fond vert, puis nous filtrons ces résultats pour ne conserver que les points nécessaires à notre application [24]. Les boîtes englobantes sont calculées automatiquement autour du corps, avec une marge de 10 %, puis les annotations sont sauvegardées au format [COCO](#) dans des fichiers [JSON](#).

Le modèle prédit donc 9 [points clés](#) :

- le cou ;
- les épaules gauche et droite ;
- les coudes gauche et droit ;
- les poignets gauche et droit ;
- les hanches gauche et droite.

Pour limiter le surapprentissage au fond vert, nous avons aussi mis en place une augmentation par changement de décor. Un masque [HSV](#) est calculé avec OpenCV pour détourer la personne, puis celle-ci est réincrustée sur plusieurs arrière-plans aléatoires. Cette étape permet d'exposer le modèle à des contextes visuels plus variés sans devoir retourner toutes les vidéos.

2.4.2. Prétraitement des images.

Les images reçues sont transformées avant l'inférence. Le code conserve le ratio de l'image en ajoutant du [padding](#) noir, aussi appelé letterbox, pour obtenir une image carrée, puis redimensionne le résultat en 256×256 , taille utilisée par la version intégrée du [Modèle 1](#). Les canaux couleur sont ensuite convertis en RGB, normalisés puis transmis au réseau. Après prédiction, les coordonnées normalisées sont reprojétées dans le repère de l'image d'origine en retirant le padding.

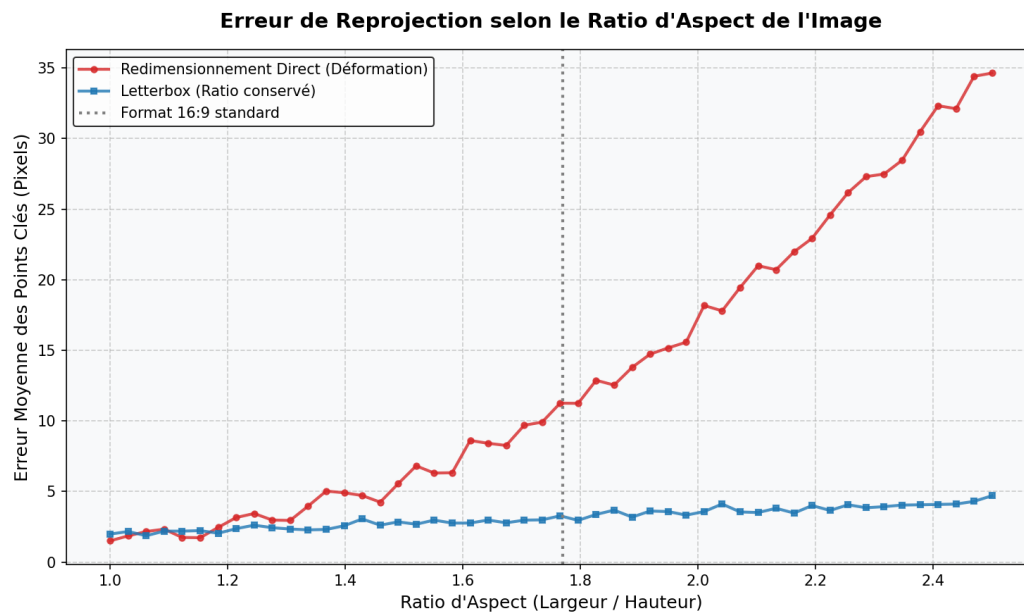


Figure 2.5 – Effet de la conservation du ratio d'aspect sur l'erreur de reprojection des points clés. Le letterbox limite la déformation géométrique par rapport à un redimensionnement direct.

2.4.3. Trace du premier modèle : architecture hybride.

Une première itération de la partie custom du [Modèle 1](#) a servi à valider la chaîne d'entraînement et les choix de représentation par [heatmaps](#). Une heatmap est une carte de probabilité : au lieu de prédire directement une coordonnée, le réseau indique les zones de l'image où un point clé a le plus de chances de se trouver. Cette version explorait un [backbone](#), c'est-à-dire la partie principale du réseau chargée d'extraire les caractéristiques de l'image avant les couches spécialisées. Ce backbone appartenait à la famille [CSPDarknet53](#), proche de l'écosystème YOLO, avec une entrée 416×416 . Il s'agit d'un réseau de neurones convolutionnel, ou [CNN](#), c'est-à-dire une architecture conçue pour extraire progressivement des formes et motifs dans une image. Elle produisait des [cartes de caractéristiques compactes](#), c'est-à-dire des représentations internes plus petites que l'image d'origine mais plus riches sémantiquement. Un facteur de réduction de 32 signifie qu'une dimension spatiale est divisée par 32 au cours du réseau : une image de 416 pixels de côté conduit donc à des cartes d'environ 13 pixels de côté avant les têtes spécialisées.

La première version du [Modèle 1](#) combinait deux sorties. Une branche auxiliaire de [heatmaps](#) utilisait une déconvolution [ConvTranspose2d](#) pour produire des cartes 26×26 , une par point clé. Ces cartes étaient entraînées avec une [loss](#), c'est-à-dire une mesure numérique de l'erreur que le modèle cherche à minimiser pendant l'apprentissage. Nous utilisons ici une [loss MSE](#) par comparaison à une gaussienne centrée sur la vérité terrain : la MSE calcule l'écart quadratique moyen entre la prédiction et la cible, ce qui pénalise fortement les grandes erreurs [25]. En parallèle, une branche de régression appliquait un [AdaptiveAvgPool1d](#) pour condenser les cartes de caractéristiques en un vecteur 1D, puis des couches linéaires prédisaient directement les coordonnées (x, y) .

La [loss](#) principale de cette version était une [OKSLoss](#). L'[OKS](#) mesure la similarité entre les points clés prédits et les points attendus. Contrairement à une [MSE](#) appliquée directement sur les coordonnées, l'[OKS](#) tient compte de l'échelle de la personne et de la tolérance associée aux différentes articulations. Cette logique est mieux adaptée à la pose humaine, car une erreur de quelques pixels n'a pas la même signification selon la taille du corps dans l'image et selon le point considéré [26].

Un problème important est apparu pendant l'entraînement : la tête linéaire produisait au départ des coordonnées non bornées, par exemple des valeurs comme 45,2 ou -12,4, alors que les cibles étaient normalisées entre 0 et 1. Comme l'[OKSLoss](#) repose sur une fonction exponentielle de la distance, cet écart massif conduisait à une loss saturée autour de 1,0 et à un gradient pratiquement nul. L'ajout d'une activation sigmoïde en sortie de la régression a borné les prédictions dans l'intervalle $[0, 1]$ et a débloqué l'apprentissage.

2.4.4. Partie custom du Modèle 1 final : architecture heatmap.

La V1 a validé la chaîne d'entraînement, mais elle présentait encore des tremblements visibles lors de l'inférence vidéo. Le diagnostic principal venait de la régression linéaire : `AdaptiveAvgPool2d` détruisait l'information spatiale 2D avant la prédiction des coordonnées, ce qui rendait le réseau moins stable frame par frame.

La version présente dans le dépôt et chargée par l'interface est implémentée dans `HmH/heatnoks/model.py`. Elle utilise [ResNet-50](#) comme [backbone](#), puis une tête de [heatmaps](#). ResNet-50 extrait efficacement des cartes de caractéristiques riches en sémantique. L'entrée étant redimensionnée en 256×256 , le réseau réduit progressivement la résolution jusqu'à des cartes 8×8 avant les têtes spécialisées. Cette réduction correspond au facteur 32 : chaque dimension spatiale est divisée par 32.

Le modèle combine deux sorties. Une tête de heatmaps, appelée `DeconvHead`, applique trois couches de déconvolution `ConvTranspose2d` successives pour remonter progressivement la résolution : 8×8 vers 16×16 , puis 32×32 , puis 64×64 . Une heatmap est produite pour chacun des 9 points clés. En parallèle, une tête de visibilité applique un `AdaptiveAvgPool2d` sur les caractéristiques du backbone pour condenser les cartes en un vecteur 1D, puis une couche linéaire prédit un score de visibilité par point clé.

La loss principale reste fondée sur l'[OKS](#). Contrairement à une [MSE](#) appliquée directement sur les coordonnées, l'[OKS](#) tient compte de l'échelle de la personne et de la tolérance associée aux différentes articulations. Cette logique est mieux adaptée à la pose humaine, car une erreur de quelques pixels n'a pas la même signification selon la taille du corps dans l'image et selon le point considéré [26]. Le choix pratique de cette loss a été appuyé par des essais complémentaires présentés dans la section 3.3.

2.4.5. Partie custom du Modèle 1 final : Spatial Softmax.

Les coordonnées sont extraites directement des heatmaps avec un [Spatial Softmax 2D](#), aussi appelé [Integral Pose Regression](#) [27]. Le principe est le suivant : nous appliquons d'abord un softmax spatial sur chaque heatmap 64×64 pour obtenir une distribution de probabilité, nous créons ensuite une grille de coordonnées (x, y) normalisées entre 0 et 1, puis nous calculons l'espérance mathématique de cette distribution. La coordonnée prédite correspond ainsi au centre de gravité de la zone activée.

Cette opération reste entièrement différentiable et permet une précision sous-pixel. Elle unifie aussi les gradients : la loss [OKS](#) agit sur les coordonnées produites par le [Spatial Softmax](#), tandis que la loss [MSE](#) continue d'encadrer la qualité des [heatmaps](#) par rapport aux gaussiennes de vérité terrain. Lors de l'inférence et de l'affichage, les points dont la visibilité est trop faible sont filtrés avec un seuil de 0,3. Ce filtrage évite d'afficher des articulations peu fiables lorsque le modèle hésite ou lorsque la partie du corps est mal détectée.

Extraction des Coordonnees

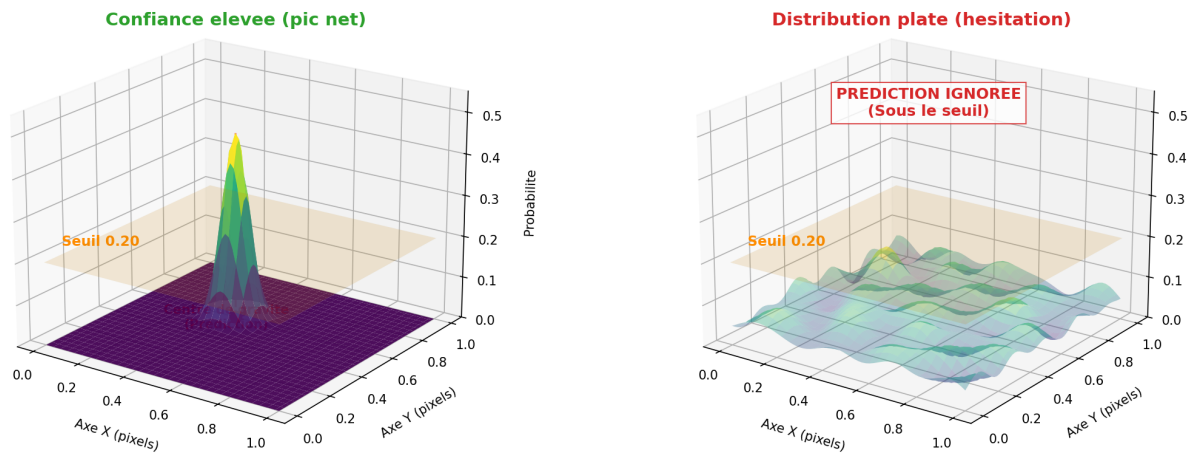


Figure 2.6 – Principe du seuil appliqué aux heatmaps : un pic net permet d’extraire une coordonnée fiable, tandis qu’une distribution plate indique une hésitation et conduit à ignorer la prédiction.

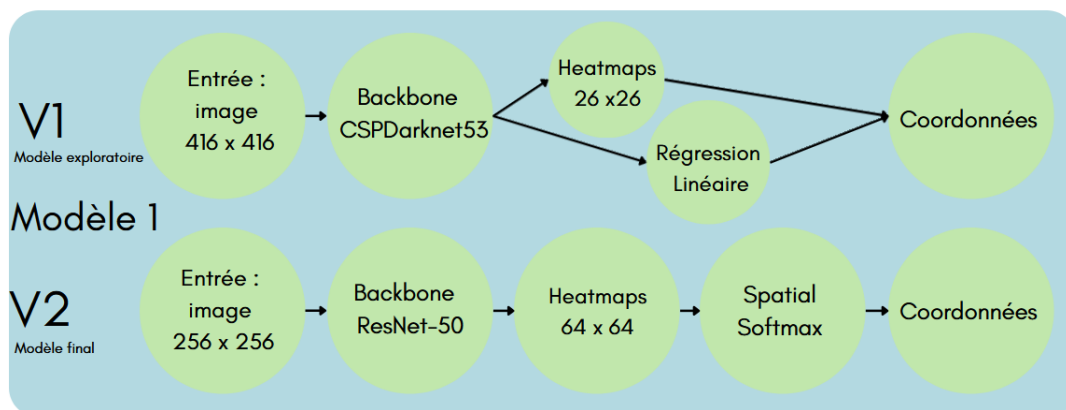


Figure 2.7 – Comparaison des deux versions de la partie custom du Modèle 1 : la V1 combine heatmaps et régression linéaire, tandis que la V2 extrait les coordonnées par Soft-Argmax.

Les poids de cette partie custom du Modèle 1 sont stockés dans le fichier `best.pt`, utilisé comme checkpoint lors de l’inférence. Dans la chaîne actuelle, `HmH/inference.py` instancie directement `HeatnoksModel`, charge ce checkpoint, puis appelle sa méthode `predict` pour obtenir les 9 points du haut du corps.

2.4.6. Détection des mains avec MediaPipe.

Pour détecter les doigts, nous utilisons [MediaPipe Hands](#). Dans la chaîne intégrée actuelle, `HmH/inference.py` exécute la détection des mains sur la frame RGB complète, en parallèle de l’inférence du haut du corps. Cette organisation simplifie l’intégration temps réel : le [Modèle 1](#) custom produit les 9 points du haut du corps pendant que [MediaPipe](#) fournit les 21 points de chaque main détectée.

Une logique de cadrage guidée par les poignets et les coudes existe aussi dans les scripts de test vidéo, notamment `inference_video.py`. Elle estime une zone probable de la main à partir du poignet, du coude et de l’épaule, puis lance [MediaPipe](#) sur ce crop. Lorsque le poignet est peu fiable, le code projette l’avant-bras dans la continuité du bras supérieur afin de conserver une zone de recherche plausible. Cette logique

sert aujourd'hui surtout aux tests isolés du module de vision : l'interface principale utilise une détection [MediaPipe](#) sur l'image complète, ce qui simplifie l'intégration temps réel.

2.5. Module de traduction.

2.5.1. Classifieur de lettres statiques.

Le classifieur de lettres statiques est documenté et entraîné dans le dossier `Static/`. La version utilisée par l'application est ensuite intégrée dans `HmH/letter_classifier.py`. Il s'agit d'un [MLP](#) de petite taille :

$$42 \rightarrow 50 \rightarrow 26$$

Cette notation signifie que le réseau reçoit 42 neurones d'entrée, passe par une couche cachée de 50 neurones, puis produit 26 neurones de sortie, un par lettre de l'alphabet. L'entrée correspond aux 21 points de la main, avec deux coordonnées par point. Avant la classification, l'axe *z* est retiré et les coordonnées sont recentrées sur le poignet, ce qui rend la représentation relative à la forme de la main plutôt qu'à sa position absolue dans l'image. La couche cachée utilise une activation [ReLU](#), une fonction qui conserve les valeurs positives et remplace les valeurs négatives par zéro afin d'introduire de la non-linéarité. La sortie produit ensuite un score pour chaque lettre.

La constitution du [dataset](#) statique repose sur des scripts de captation et de transformation du dossier `Static/`. Les vidéos ou images de signes sont analysées avec [MediaPipe](#), puis le script `tris.py` applique ce prétraitement et sépare strictement les données en deux ensembles : 80 % pour l'entraînement et 20 % pour l'évaluation. Ce découpage évite de mesurer le modèle sur les mêmes exemples que ceux utilisés pendant l'apprentissage. Il limite donc le risque de *data leakage*, c'est-à-dire une fuite d'information entre entraînement et évaluation.

Une première version de l'entraînement avait été codée manuellement, avec les calculs de rétropropagation explicitement écrits. Cette approche était utile pour comprendre les mathématiques du réseau, mais elle était lente et difficile à faire évoluer. Nous avons ensuite migré l'entraînement vers [PyTorch](#), dans `training_optimized.py`, avec `nn.Module` et un optimiseur `SGD`. [PyTorch](#) est donc utilisé pour la phase d'apprentissage, de recherche d'[hyperparamètres](#) et d'export des poids, pas comme dépendance centrale du MLP en production. Cette migration a fortement accéléré les essais et permis de tester plusieurs configurations. Le choix des hyperparamètres a été réalisé à partir d'une recherche contrôlée sur le taux d'apprentissage, la taille de la couche cachée, le nombre d'itérations et la fonction d'activation, avec une évaluation sur un jeu séparé. La logique complète de cette recherche est détaillée dans les travaux complémentaires, section 3.2. La meilleure configuration retenue pour le classifieur statique utilise 50 neurones cachés, une activation [ReLU](#), 50 000 itérations et un taux d'apprentissage de 0,9. Elle atteint 90,42 % de précision sur le jeu d'évaluation, avec une *loss* finale de 0,0058.

Ce taux d'apprentissage élevé pourrait sembler risqué avec une configuration classique fondée sur `CrossEntropyLoss`, une *loss* spécialement adaptée aux problèmes de classification où une seule classe correcte doit être choisie. Dans notre cas, l'association entre une sortie convertie par softmax et une *loss* de type [MSE](#) a joué un rôle d'amortisseur : le softmax transforme les scores de sortie en probabilités positives dont la somme vaut 1, et les gradients restent alors moins violents qu'avec des [logits](#) bruts. Les logits sont les scores non normalisés produits avant conversion en probabilités. Cette configuration n'est pas nécessairement la plus générale, mais elle s'est révélée efficace sur notre [dataset](#) statique. Elle illustre aussi l'intérêt d'évaluer empiriquement plusieurs choix d'entraînement plutôt que de supposer qu'une configuration théoriquement standard sera automatiquement meilleure.

Les poids du réseau sont chargés depuis quatre fichiers JSON :

- `W1.json` de dimension 42×50 ;

- `b1.json` de dimension 1×50 ;
- `W2.json` de dimension 50×26 ;
- `b2.json` de dimension 1×26 .

Ce choix d'export rend l'inférence légère. Pendant l'entraînement, `PyTorch` gère les gradients et l'optimisation. Une fois le modèle sélectionné, seules les matrices utiles sont extraites, transposées si nécessaire, puis sauvegardées en `JSON`. En production, le MLP ne recharge pas `PyTorch` : `letter_classifier.py` exécute directement les mathématiques avec `NumPy`. Nous avons donc recodé explicitement les briques nécessaires : chargement des matrices `JSON`, produit matriciel, `ReLU` via `np.maximum`, softmax stable et choix de la classe maximale.

```
1 def _softmax(z):
2     z = z - np.max(z, axis=-1, keepdims=True)
3     exp = np.exp(z)
4     return exp / np.sum(exp, axis=-1, keepdims=True)
5
6 def predict(self, features):
7     x = np.asarray(features, dtype=float).reshape(1, -1)
8     h = np.maximum(0.0, x @ self.W1 + self.b1)
9     out = _softmax(h @ self.W2 + self.b2)[0]
10    idx = int(np.argmax(out))
11    return ALPHABET[idx], float(out[idx])
```

Listing 2.1 – Fonctions essentielles recodées pour l'inférence légère du MLP.

Cette séparation entre entraînement et production est importante. Elle permet d'utiliser `PyTorch` là où il est utile, c'est-à-dire pour calculer rapidement les gradients et ajuster les poids, tout en gardant un classifieur embarqué très simple au moment de la démonstration. Le serveur n'a alors besoin que des fichiers de poids et de `NumPy` pour reconnaître une lettre statique.

Le classifieur est donc prévu pour les 26 lettres. En observation réelle, certaines lettres restent toutefois difficiles à séparer avec une seule image. Les confusions les plus importantes concernent surtout J/Y, P/K et D/Z : J et Y deviennent proches sans trajectoire, tandis que P/K et D/Z dépendent fortement de l'orientation de la main et de la qualité de détection. La lettre Q implique aussi un mouvement, mais son orientation est beaucoup plus distinctive dans notre jeu d'essai ; elle ne fait donc pas apparaître les mêmes difficultés dans les tests du MLP. Ces confusions justifient l'ajout d'un modèle dynamique : pour certaines lettres, la trajectoire apporte une information que le MLP statique ne peut pas exploiter seul.

2.5.2. Évaluation du MLP statique.

Afin d'évaluer plus finement le comportement du classifieur statique, nous utilisons une matrice de confusion. Cette matrice croise, pour chaque exemple, la vraie lettre et la lettre prédite ; elle permet donc de voir non seulement combien d'erreurs sont commises, mais aussi quelles lettres sont confondues entre elles. L'accuracy globale, c'est-à-dire le pourcentage total de prédictions correctes, donne une première indication, mais elle ne suffit pas à juger la qualité du modèle, surtout lorsque les résultats en conditions réelles peuvent différer des résultats obtenus sur un jeu de validation contrôlé.

Le `dataset` statique est constitué à partir d'environ 30 Go de vidéos où chaque signe de l'alphabet est réalisé sous forme de posture isolée. Ces vidéos sont ensuite transformées en images puis en coordonnées de main exploitables par le MLP. Ce volume permet de tester le fonctionnement du classifieur sur des configurations maîtrisées, mais il ne couvre pas toutes les variations possibles d'orientation, de distance à la caméra ou de morphologie de main. Cette limite explique pourquoi l'évaluation par matrice de confusion doit être complétée par des essais devant l'interface.

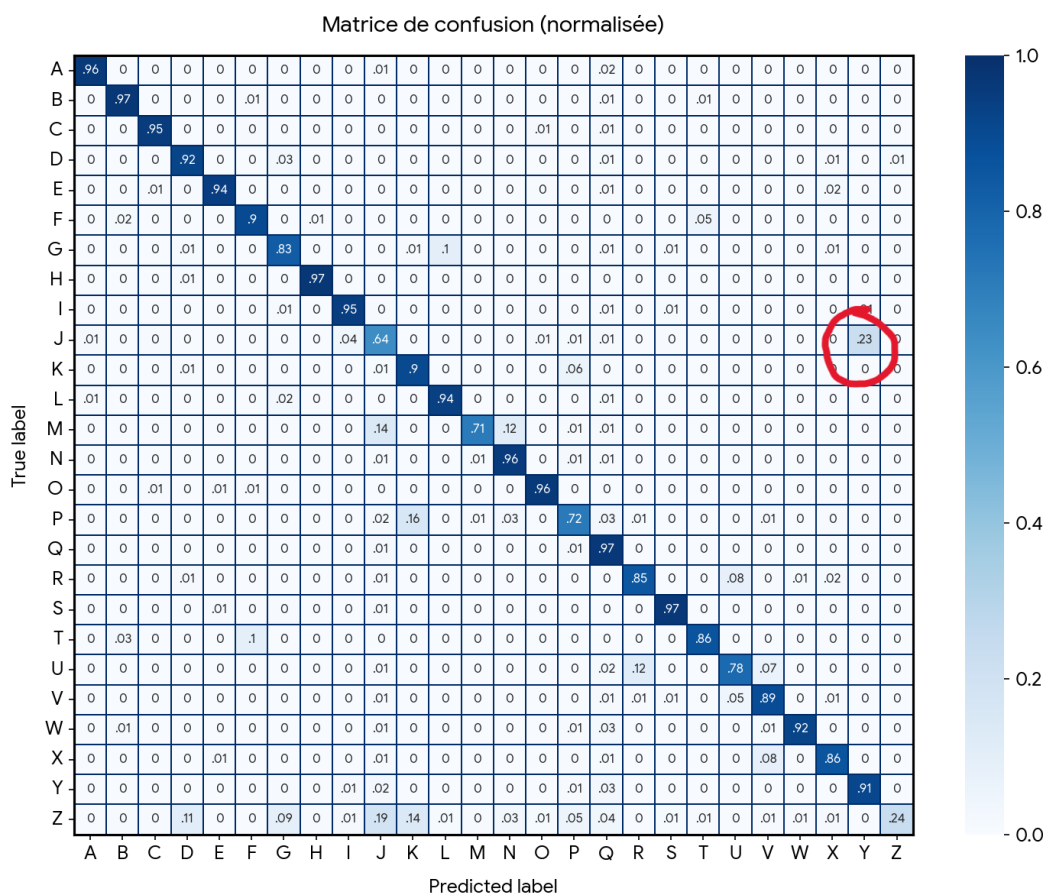


Figure 2.8 – Matrice de confusion du MLP sur le jeu de validation. Elle permet d'identifier les lettres statiques les plus souvent confondues.



(a) Lettre J



(b) Lettre Y

Figure 2.9 – Comparaison entre J et Y : sans prise en compte du mouvement, le MLP observe une posture très proche pour ces deux lettres.

La matrice de confusion du MLP confirme que les erreurs se concentrent surtout sur des lettres visuellement proches ou sur des lettres nécessitant un mouvement. Elle rend ces erreurs beaucoup plus lisibles qu'un simple fichier JSON d'évaluation, car elle montre directement les couples de lettres les plus problématiques. Les confusions les plus significatives concernent J/Y, P/K et D/Z. La confusion entre J et Y est particulièrement prévisible : si l'on ne tient compte que d'une image fixe, ces deux lettres peuvent présenter une configuration de main presque identique. Pour P/K et D/Z, l'erreur vient davantage de l'orientation de la main et de la précision des points détectés. À l'inverse, Q reste mieux séparée dans

nos essais, car son orientation est spécifique par rapport aux autres lettres du corpus. Cette limite justifie l'ajout d'une branche temporelle, surtout pour les lettres dont le tracé ou l'orientation évolue pendant le geste.

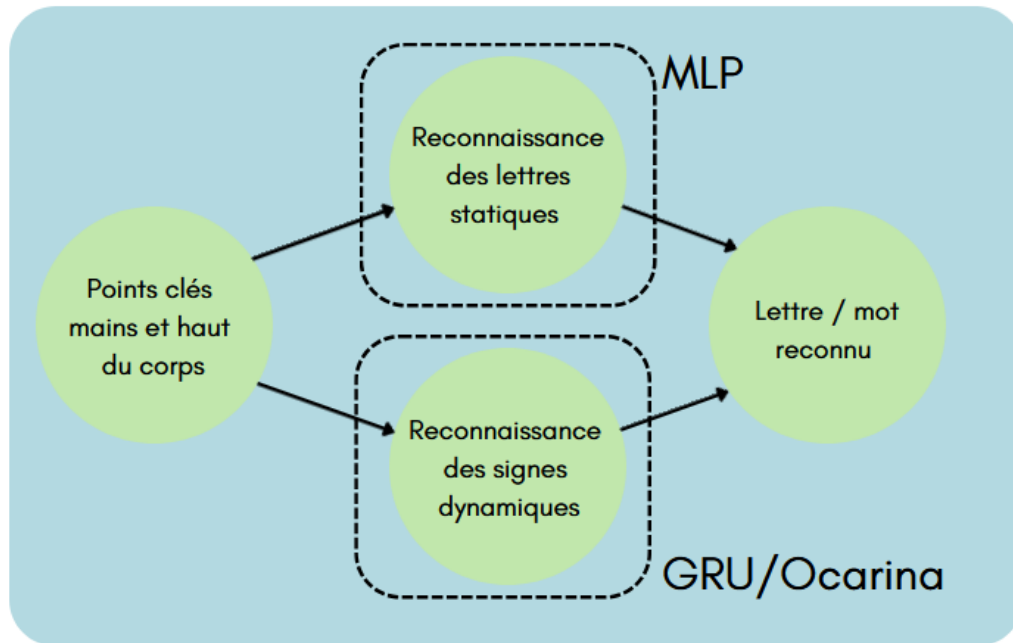


Figure 2.10 – Séparation du Modèle 2 : les points clés peuvent être traités par le MLP pour une posture statique ou par le GRU lorsqu'une séquence temporelle est nécessaire.

2.5.3. Modèle 2 temporel : GRU.

Pour traiter les lettres dynamiques, nous avons ajouté une branche temporelle dans le Modèle 2. Nommé Modèle *Ocarina*, situé dans `Modèle_Ocarina`. Contrairement au [MLP](#), qui classe une seule configuration de main, cette branche reçoit une séquence de frames et apprend l'évolution du geste dans le temps et l'espace. Ce choix répond directement au problème posé par les lettres qui ne peuvent pas être reconnues à partir d'une posture isolée.

Le modèle est implémenté dans `Ocarina_GRU.py`. Il repose sur un réseau récurrent de type [GRU](#), adapté aux séquences car il conserve une mémoire interne tout en restant plus léger qu'une architecture récurrente plus complexe de type [LSTM](#) [28]. Le [GRU](#) utilise des mécanismes de portes pour sélectionner l'information à conserver ou à oublier au fil des frames [29]. En complément des articles scientifiques, nous avons utilisé des ressources vidéo pour comprendre plus rapidement le rôle des portes et la différence pratique entre [LSTM](#) et [GRU](#) [30]. Cette propriété est importante pour la LSF : le début, la trajectoire et la fin d'un geste peuvent tous contribuer à la classe finale.

Dans la version actuelle du modèle, chaque frame est représentée par 42 valeurs prétraitées correspondant aux 21 points de la main en deux dimensions. Trois étapes de normalisation sont appliquées avant l'envoi au réseau. Les coordonnées brutes sont d'abord ramenées dans l'intervalle $[0, 1]$ en divisant par la largeur et la hauteur de l'image. Cette étape est essentielle pour assurer la cohérence entre l'entraînement, qui utilise des coordonnées issues de vidéos en 1920×1080 , et l'inférence, où [MediaPipe](#) fournit directement des coordonnées normalisées. Sans cette harmonisation, le modèle est entraîné dans un espace de coordonnées différent de celui de la démonstration, ce qui dégrade fortement la reconnaissance en conditions réelles. Les points sont ensuite recentrés sur le poignet, puis divisés par la distance entre le

poignet et l'articulation métacarpienne du majeur. Cette mise à l'échelle par la taille de la main supprime la dépendance à la distance entre la personne et la caméra, qui faisait auparavant varier l'amplitude des coordonnées d'un facteur deux ou trois entre deux prises. Une dernière étape miroir les coordonnées horizontalement si la main détectée n'est pas du côté de référence choisi, ce qui rend le modèle indépendant du côté utilisé par le signeur.

L'entrée est ensuite projetée par un bloc d'embedding :

$$42 \rightarrow 128 \rightarrow 96$$

Ce bloc combine deux couches linéaires, des activations **ReLU** et un **dropout** de 0,3. La séquence transformée est transmise à un **GRU** bidirectionnel à deux couches, avec une taille cachée de 96. Le caractère bidirectionnel signifie que la séquence est lue à la fois du début vers la fin et de la fin vers le début, puis que les deux représentations sont concaténées. Cette propriété est utile pour les lettres dynamiques, car le sens d'un geste comme J ou Z peut être confirmé en regardant à la fois la posture initiale et la trajectoire finale. La sortie utilisée pour la classification n'est plus la mémoire de la dernière frame, mais une moyenne masquée sur l'ensemble des frames réelles de la séquence. Ce changement est important : les séquences sont complétées par des zéros pour atteindre une taille fixe, et la dernière frame est donc très souvent du **padding**. L'ancienne version, qui prenait uniquement la dernière frame, forçait le réseau à apprendre à ignorer ces zéros, ce qui consommait inutilement de la capacité du modèle. La moyenne masquée n'agrège que les frames porteuses d'information, à l'aide d'un masque binaire transmis avec la séquence. Une normalisation par couche et un **dropout** supplémentaire sont appliqués avant le classifieur linéaire final, qui produit les **logits** des 26 classes.

2.5.4. Dataset et entraînement du GRU.

Le chargement des données est géré par `Dataset.py`. Le **dataset** attendu est un dossier contenant des fichiers **JSON** au format **COCO** adapté, comme ceux produits par notre chaîne de segmentation. Un exemple simplifié de cette structure est présenté en annexe A. Pour chaque fichier, le script trie les images par `frame_index`, récupère l'annotation associée à chaque frame, extrait les coordonnées (x, y) et remplace les frames sans annotation par un vecteur nul. Les labels sont déduits du nom des fichiers, ce qui impose une convention de nommage stable dans le **dataset**. Le `Dataset` retourne désormais un triplet (`frames`, `masque`, `label`), où le masque indique pour chaque frame si elle contient une vraie observation ou un **padding**.

Les séquences sont normalisées en longueur à 45 frames, valeur alignée sur celle utilisée par l'intégration web. Cette harmonisation supprime un écart présent dans la version précédente, qui entraînait le modèle sur 50 frames mais le nourrissait avec 45 frames en production. Si une séquence est plus courte que 45, elle est complétée par des zéros et son masque marque ces frames comme **padding**. Si elle est plus longue, ce sont les 45 dernières frames qui sont conservées, car la fin d'un signe porte généralement l'information la plus discriminante.

La constitution du split d'entraînement et de validation a aussi été corrigée. La version précédente créait deux instances du `Dataset` et appliquait à chacune la même permutation aléatoire d'indices, en supposant que l'ordre des fichiers serait identique entre les deux instances. Or chaque instance exécute son propre `os.listdir`, dont l'ordre n'est pas garanti d'une instance à l'autre. Cette hypothèse implicite produisait, dans certains cas, un recouvrement partiel entre l'entraînement et la validation, ce qui faussait fortement les mesures et expliquait l'écart entre la loss d'entraînement, très basse, et la loss de validation, dix fois supérieure. La nouvelle implémentation construit une liste unique de fichiers triée par ordre alphabétique, applique une permutation contrôlée par une graine fixe, puis dérive de cette permutation l'ensemble d'entraînement et l'ensemble de validation. Une vérification ajoutée s'assure qu'aucun chemin de fichier n'appartient simultanément aux deux ensembles.

Le script `Train.py` entraîne le GRU avec PyTorch [31]. Les hyperparamètres retenus dans le code sont une taille de `batch` de 32, un maximum de 200 époques, un taux d'apprentissage initial de 1×10^{-3} et une séparation 80 % entraînement / 20 % validation. L'optimiseur utilisé est AdamW, qui ajoute une régularisation par décroissance des poids à Adam, avec un coefficient de 1×10^{-4} . Le taux d'apprentissage suit une décroissance en cosinus sur la durée totale de l'entraînement. La loss utilisée est `CrossEntropyLoss` avec un lissage de label de 0,05, ce qui évite que le réseau ne devienne excessivement confiant sur les exemples d'entraînement. Des poids de classe sont calculés à partir des fréquences observées et passés à la loss pour compenser les déséquilibres entre lettres. Les gradients sont écrêtés à une norme maximale de 1,0 avant chaque pas d'optimisation. Un arrêt anticipé interrompt l'entraînement si la précision de validation ne progresse plus pendant 40 époques consécutives, et seuls les poids correspondant à la meilleure époque de validation sont sauvegardés.

L'augmentation appliquée pendant le chargement combine quatre transformations. Un bruit gaussien d'écart-type 0,01 est ajouté aux coordonnées normalisées pour simuler les imprécisions de détection. Une occlusion masque environ 5 % des coordonnées de manière aléatoire pour rendre le modèle robuste aux landmarks manquantes. Une rotation aléatoire de plus ou moins 15 degrés est appliquée à la séquence entière pour simuler les variations d'inclinaison du poignet. Enfin, un facteur d'échelle aléatoire de plus ou moins 10 % perturbe la taille apparente du geste. Les deux dernières augmentations sont plus efficaces qu'un simple bruit par coordonnée, car elles produisent des perturbations cohérentes au niveau du geste plutôt qu'au niveau du pixel.

À la fin de l'entraînement, le script sauvegarde les poids dans `ocarina_gru_v2.pth` et un fichier `ocarina_classes.json` listant l'ordre des classes. Il est ensuite utilisé par `Diagnosis.py`. Afin de générer des figures et `training_curves.png`, qui synthétise l'évolution de la loss d'entraînement, de la loss de validation et de l'accuracy de validation. Sur notre entraînement de référence, mené sur 799 séquences réparties en 639 vidéos d'entraînement et 160 vidéos de validation, la précision de validation atteint 91,88 % à l'époque 47. L'écart entre la loss d'entraînement, mesurée à 0,56 à ce point, et la loss de validation, mesurée à 0,66, reste limité, ce qui contraste fortement avec l'écart d'un facteur dix observé sur la version précédente du modèle et confirme la disparition du défaut de séparation entre les ensembles. L'arrêt anticipé interrompt l'entraînement à l'époque 87, sans amélioration significative au-delà de l'époque 47.

Le GRU est intégré à la chaîne principale. Le serveur charge automatiquement les poids Ocarina au démarrage et expose une route `hearmyhands.asia/sign_predict` pour la prédiction temporelle en mode séparé. Dans le mode intégré, l'application web appelle directement le même moteur d'inférence. La séquence envoyée au modèle dynamique n'est pas construite à partir de toute la vidéo en continu : côté navigateur, un seuil visuel déclenche l'envoi lorsque la main entre dans la zone de signature, puis le serveur normalise les points de main avant la prédiction, en appliquant la même chaîne de centrage, de mise à l'échelle et de mirror que celle utilisée à l'entraînement. Cette parité entre les deux phases est explicitement assurée par l'appel d'une fonction commune `normalize_frame`, partagée entre `Dataset.py` et le script d'inférence.

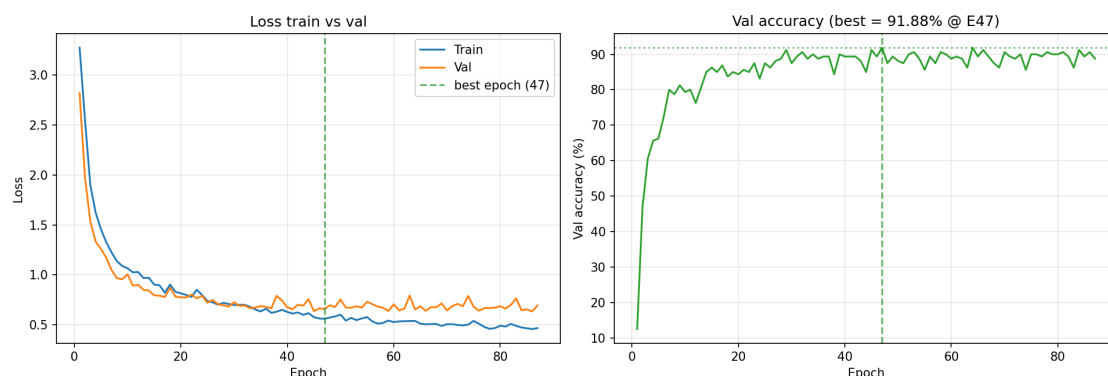


Figure 2.11 – Courbes d’entraînement du GRU Ocarina : évolution des loss d’entraînement et de validation, ainsi que de la précision de validation. La proximité entre les deux loss tout au long de l’entraînement confirme la disparition du défaut de séparation entre les ensembles présent dans la version précédente.

La lecture de ces courbes montre une convergence rapide : la précision de validation progresse fortement pendant les premières dizaines d’époques, puis se stabilise autour du palier maximal atteint à l’époque 47. Contrairement à la version précédente, où la loss de validation oscillait à un niveau dix fois supérieur à la loss d’entraînement, les deux courbes restent proches sur l’ensemble de l’entraînement. Ce comportement traduit une bonne capacité de généralisation sur le jeu de validation, à confronter cependant aux essais en conditions réelles pour mesurer la robustesse pratique du système.

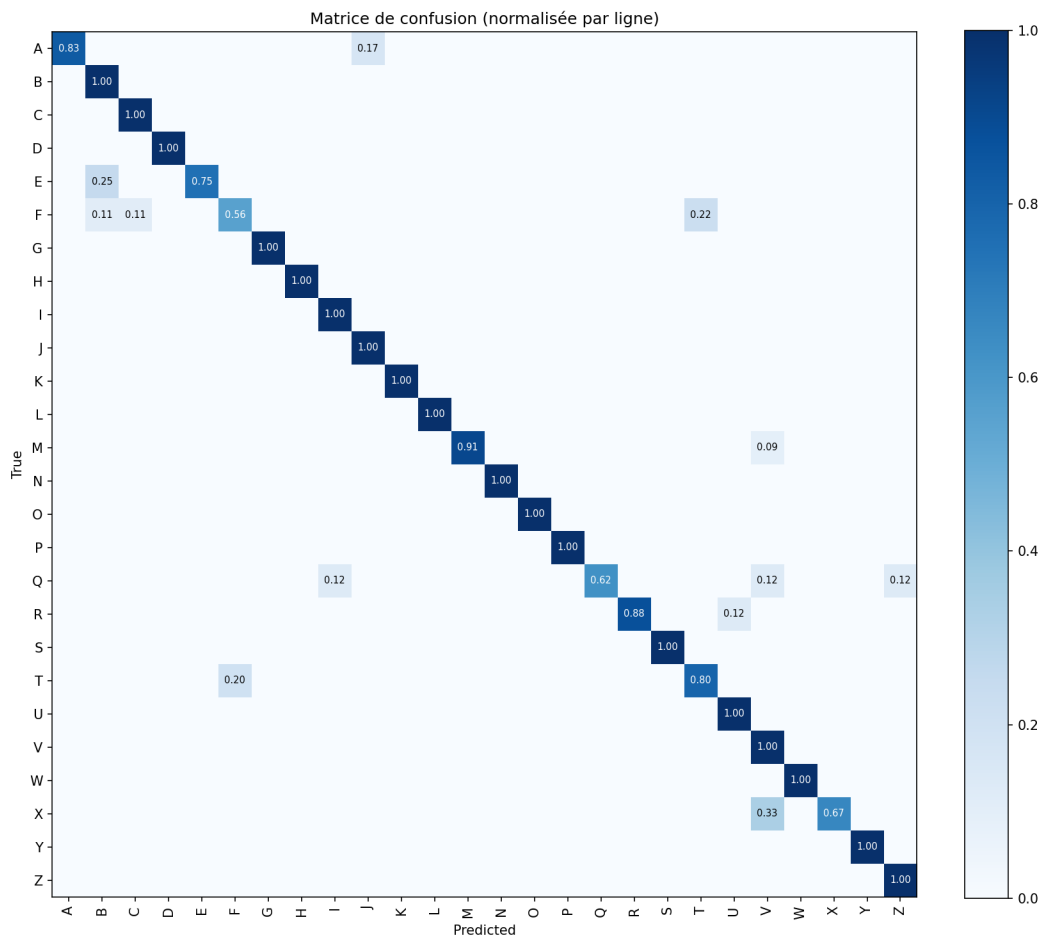


Figure 2.12 – Matrice de confusion du GRU Ocarina sur le jeu de validation. Elle complète l’accuracy globale de 91,88 % en montrant les erreurs lettre par lettre.

La matrice de confusion montre une diagonale majoritairement nette, ce qui confirme que le modèle reconnaît correctement la plupart des lettres dans le jeu de validation. Les erreurs restantes sont surtout localisées entre quelques lettres proches. On observe par exemple une confusion entre F et T, deux configurations où la position relative du pouce et de l’index peut être difficile à distinguer lorsque la main est légèrement tournée ou lorsque certains points sont bruités. Cette lecture est utile pour orienter les prochaines collectes : il ne suffit pas d’augmenter uniformément le [dataset](#), il faut aussi renforcer les exemples des paires de lettres proches.

Les essais en conditions réelles donnent cependant des résultats plus irréguliers que la matrice de confusion. L’intégration du modèle dynamique reste plus sensible que celle du [MLP](#), car elle dépend à la fois de la segmentation temporelle du flux vidéo, de la stabilité des [keypoints](#) et de la manière dont l’utilisateur commence puis termine son geste. Le [dataset](#) temporel contient environ quarante vidéos par lettre, ce qui reste probablement faible pour généraliser à des gestes réalisés par des personnes peu expérimentées en [LSF](#). Une partie des variations vient aussi de notre propre apprentissage des signes : deux prises d’une même lettre peuvent différer dans l’amplitude, le rythme ou l’orientation. Ces constats ne remettent pas en cause l’intérêt de la branche [GRU](#), mais ils montrent que son usage en démonstration demande une stabilisation plus fine et une collecte plus représentative.



Figure 2.13 – Exemple de lettres proches pouvant expliquer certaines confusions résiduelles du modèle temporel.

2.5.5. Trace de la première version du GRU.

La première version du modèle temporel a permis de valider la chaîne d'entraînement, mais elle présentait plusieurs limites que les essais en conditions réelles ont rendues visibles. Cette sous-section décrit ces limites et les corrections apportées dans la version actuelle ; chaque choix technique présenté dans les sous-sections suivantes trouve son origine dans l'un de ces constats.

Architecture initiale. La V1 reposait sur un GRU unidirectionnel à une seule couche, avec une taille cachée de 64 et un bloc d'embedding $42 \rightarrow 256 \rightarrow 64$. La sortie utilisée pour la classification était la mémoire associée à la dernière frame de la séquence. Cette architecture, simple et rapide à entraîner, a permis d'obtenir un premier modèle fonctionnel et de valider l'ensemble du pipeline.

Limite 1 : décalage entre l'espace de coordonnées d'entraînement et celui de l'inférence. Les fichiers JSON du [dataset](#) contiennent des coordonnées exprimées en pixels dans le repère vidéo d'origine, généralement 1920×1080 . La V1 utilisait ces coordonnées directement après un simple recentrage sur le poignet. À l'inférence, en revanche, [MediaPipe](#) fournit des coordonnées déjà normalisées dans l'intervalle $[0, 1]$. Le modèle était donc entraîné sur un espace de coordonnées différent de celui de la démonstration, ce qui dégradait fortement la reconnaissance dès la sortie du jeu de validation. La version actuelle ramène systématiquement les coordonnées dans $[0, 1]$ avant tout traitement, ce qui garantit que l'entraînement et l'inférence partagent la même distribution de valeurs.

Limite 2 : sensibilité à la distance et au côté du signeur. Le simple recentrage sur le poignet ne supprimait ni la variation d'échelle liée à la distance entre la personne et la caméra, ni la différence entre une main gauche et une main droite. Une même lettre signée par deux personnes pouvait produire des séquences numériquement très différentes. La version actuelle ajoute deux étapes de normalisation : une mise à l'échelle par la taille de la main, mesurée comme la distance entre le poignet et l'articulation métacarpienne du majeur, et un *mirror* horizontal des coordonnées lorsque la main détectée n'est pas du côté de référence choisi. Ces deux étapes rendent le modèle indépendant de la distance à la caméra et du côté utilisé par le signeur.

Limite 3 : exploitation incomplète de la séquence. La V1 prenait uniquement la mémoire de la dernière frame pour la classification. Or les séquences sont complétées par des zéros pour atteindre une taille fixe, et la dernière frame correspond donc très souvent à du [padding](#). Le réseau devait apprendre à ignorer cette frame de bourrage, ce qui consommait inutilement de la capacité du modèle. La version actuelle applique une moyenne masquée sur l'ensemble des frames réelles de la séquence, à l'aide d'un masque binaire transmis avec les données. Toutes les frames porteuses d'information contribuent ainsi à la prédiction.

Limite 4 : capacité limitée pour les lettres dynamiques. Une seule couche de GRU unidirectionnel s'est avérée insuffisante pour exploiter pleinement la dynamique de gestes comme J ou Z, où le sens du mouvement compte autant que la posture finale. La version actuelle utilise un GRU bidirectionnel à deux couches, avec une taille cachée de 96. Le caractère bidirectionnel signifie que la séquence est lue à la fois du début vers la fin et de la fin vers le début, puis que les deux représentations sont concaténées : la posture initiale et la trajectoire finale contribuent ainsi conjointement à la classification.

Limite 5 : fuite entre entraînement et validation. Le script d'entraînement de la V1 créait deux instances séparées du Dataset et appliquait à chacune la même permutation aléatoire d'indices, en supposant que l'ordre des fichiers serait identique entre les deux instances. Or chaque instance exécute son propre `os.listdir`, dont l'ordre n'est pas garanti d'une instance à l'autre. Cette hypothèse implicite produisait un recouvrement partiel entre les deux ensembles, ce qui faussait fortement les mesures et expliquait l'écart d'un facteur dix observé entre la loss d'entraînement et la loss de validation. La version actuelle construit la liste des fichiers une seule fois, triée par ordre alphabétique, puis applique une permutation contrôlée par une graine fixe avant de séparer entraînement et validation. Une vérification explicite garantit qu'aucun fichier n'appartient simultanément aux deux ensembles.

Ces cinq corrections expliquent à la fois la nouvelle architecture détaillée dans la sous-section précédentes et le réalisme accru des métriques de validation présentées en §2.5.3.

2.6. Hypothèses retenues.

Nous avons explicité les principales hypothèses qui cadrent notre démarche, car elles conditionnent à la fois le périmètre du projet, les choix techniques et l'interprétation des résultats.

Hypothèse	Justification	Risque associé
Périmètre maîtrisé	Valider d'abord une chaîne complète permet de garder un objectif mesurable.	Le passage à des échanges linguistiques complets demandera d'autres données et d'autres méthodes.
Webcam standard	Le matériel côté utilisateur doit rester courant.	La qualité de détection dépend fortement de l'environnement.
Architecture client-serveur	Séparer interface et modèle rend le système plus lisible et facilite le déploiement.	La latence réseau peut influencer la fluidité.
Représentation par squelette	Les coordonnées réduisent la dimension et simplifient la classification.	Une erreur de détection des points se propage au classifieur.
MLP pour le statique	Les lettres fixes peuvent être classées à partir d'une configuration de main.	Confusions possibles entre lettres visuellement proches.

Table 2.1 – Hypothèses structurantes du développement.

2.7. Constitution et usage du dataset.

2.7.1. Contraintes de collecte.

Le **dataset** est un élément central du projet. Sans données variées, un modèle peut sembler performant sur quelques exemples mais échouer dès qu'un nouvel utilisateur signe différemment. Nous avons donc défini un protocole de captation visant à limiter les variations inutiles tout en conservant une diversité suffisante. Pour le **MLP**, nous avons filmé des signes fixes de l'alphabet afin d'obtenir des configurations de main isolées. Pour le **GRU**, nous avons filmé des prises vidéo plus longues afin de conserver la trajectoire des lettres dynamiques. Le pôle communication a joué un rôle important dans cette étape : en plus des échanges avec les professeurs et les associations, il a participé à la collecte et à l'organisation de données utiles aux deux branches du **Modèle 2**.

Les vidéos sont prises en intérieur, avec un éclairage frontal, un arrière-plan simple et un cadrage allant approximativement de la tête à la taille. Les vêtements doivent contraster avec la peau et éviter les bijoux ou éléments visuels pouvant perturber la détection. Ces contraintes améliorent la qualité des points extraits par le module de vision.

2.7.2. Données utilisées par le système.

La chaîne d'**inférence** utilise plusieurs artefacts complémentaires. La partie custom du **Modèle 1** charge le checkpoint `best.pt`. La détection des mains repose sur le modèle `MediaPipe hand_landmarker.task`. Le **MLP** du **Modèle 2** charge ses poids au format **JSON** depuis le dossier `HmH/Poids/`. Enfin, le classifieur temporel charge les poids `ocarina_gru_v2.pth` depuis `Modèle_Ocarina/`. Le système combine donc les artefacts des deux modules de traduction, statique et dynamique, dans une même chaîne d'exécution.

Dans la chaîne actuelle, les données utilisées en **inférence** ne sont pas des images brutes conservées par l'application. Le navigateur envoie une image, l'**API** extrait les **keypoints**, puis renvoie uniquement les coordonnées et la prédiction. Cette organisation limite la quantité de données manipulées côté interface.

2.7.3. Outils complémentaires autour du dataset.

Deux outils complémentaires ont accompagné la constitution du **dataset**. Le premier est l'outil de segmentation vidéo. Il est surtout lié au **GRU**, car le modèle temporel a besoin de clips courts, propres et alignés plutôt que de vidéos longues difficiles à exploiter. Le second est l'exploration **Blender**, qui sert à tester la génération synthétique d'images annotées. Ces travaux sont détaillés dans les résultats complémentaires, car ils soutiennent le projet sans constituer le coeur de la chaîne de démonstration.

2.8. Mise en œuvre expérimentale.

2.8.1. Environnement logiciel.

Le développement est réalisé principalement en Python et **JavaScript**. Côté serveur, nous utilisons **Flask** pour l'application web, **Flask-SocketIO** pour le transport temps réel, **Gunicorn** pour l'exécution en production, **PyTorch** et **TorchVision** pour la partie custom du **Modèle 1** [31], **OpenCV** pour le traitement d'image, **NumPy** pour les calculs numériques, **MediaPipe** pour la partie mains du **Modèle 1**, et **psutil** pour la page de monitoring. Le module **Requests** reste utilisé lorsque l'on active le mode **API** séparée. Côté client, l'interface repose sur **HTML**, **CSS**, **JavaScript** et **Socket.IO**. Les outils de **dataset** utilisent également **Flask**, **OpenCV**, **MediaPipe** et **Blender** afin de couvrir la segmentation, l'annotation, la visualisation et la génération synthétique.

Les extraits de code complets ne sont pas intégrés dans le corps du rapport afin de préserver la lisibilité. Ils sont disponibles dans le dépôt GitHub principal du projet, référencé en annexe C.

L'application web fonctionne par défaut sur le port 5000 et charge les modèles dans le même processus. Si le mode séparé est activé avec `USE_HTTP_MODEL=1`, l'API modèle fonctionne sur le port 5001 et les variables d'environnement `MODEL_API_URL`, `SIGN_API_URL`, `MODEL_TIMEOUT`, `SIGN_TIMEOUT` et `PORT` permettent d'adapter le déploiement. La version accessible en ligne est hébergée sur [Google Cloud](https://hearmyhands.asia/) à l'adresse `https://hearmyhands.asia/`. Cette architecture centralise l'inférence, simplifie les mises à jour du modèle, permet l'accès depuis téléphone ou ordinateur et facilite les démonstrations à distance.

2.8.2. Gestion des erreurs.

La gestion des erreurs n'a pas seulement un rôle de confort utilisateur. Elle permet aussi de sécuriser les essais expérimentaux : lorsqu'une image est invalide, qu'un modèle ne répond pas ou qu'un résultat est vide, l'interface reste exploitable et le problème peut être isolé sans interrompre toute la chaîne. Cette robustesse facilite les démonstrations, les tests comparatifs et l'identification précise des briques à corriger.

2.8.3. Scripts d'inférence.

Deux scripts complètent l'application web. `inference_heatnoks.py` permet de tester le [Modèle 1](#) sur une image et de sauvegarder une visualisation des points prédits. `inference_video.py` permet d'exécuter l'inférence sur une webcam ou une vidéo, avec affichage OpenCV du [squelette](#) et des mains. Ces scripts sont utiles pour isoler les performances du module de vision sans passer par l'interface web.

2.9. Résultats et analyse.

2.9.1. Fonctionnement validé par le code.

Les fonctionnalités principales ont d'abord été testées séparément : capture webcam, transport des frames, détection du haut du corps, détection des mains, classification par [MLP](#), classification temporelle par [GRU](#) et affichage dans l'interface. Nous avons ensuite intégré ces briques progressivement, pièce par pièce, jusqu'à obtenir une chaîne fonctionnelle pour la reconnaissance de lettres statiques et dynamiques. Dans cette chaîne, la webcam est activée dans le navigateur, les frames sont compressées et envoyées au serveur lorsque le seuil de capture est franchi, puis le [Modèle 1](#) extrait les points du haut du corps et les articulations des mains. Ensuite, selon le mode choisi dans l'interface, le [Modèle 2](#) utilise soit le [MLP](#) pour reconnaître une lettre statique à partir d'une main isolée, soit le [GRU](#) pour exploiter une séquence temporelle de [keypoints](#). L'interface affiche alors le [squelette](#), les mains, la prédiction courante et la suite de lettres validées.

Le système n'est pas seulement un script isolé. Il relie une interface utilisateur, un transport temps réel, un moteur d'inférence, un réseau de pose, un détecteur de mains, un classifieur de lettres statiques et un classifieur temporel. Cette intégration constitue un résultat technique important, car elle démontre la cohérence de l'architecture globale.

2.9.2. Analyse critique.

La chaîne actuelle est robuste dans sa conception, mais elle reste dépendante de plusieurs conditions. La qualité de la webcam, l'éclairage, le cadrage et la visibilité des mains influencent directement la qualité des [keypoints](#). Le classifieur de lettres ne peut pas corriger une main mal détectée : il ne reçoit que les coordonnées produites par le module de vision.

Les deux branches du **Modèle 2** répondent à des besoins complémentaires. Chacune présente des avantages utiles pour le démonstrateur, mais aussi des limites qui expliquent pourquoi nous avons conservé deux modes de reconnaissance.

Modèle	Points forts	Limites observées
MLP statique	Rapide, léger, adapté aux lettres fixes, faible coût de calcul.	Sensible à l'orientation de la main et incapable d'exploiter le mouvement.
GRU temporel	Exploite la trajectoire du geste, mieux adapté aux lettres dynamiques.	Demande des séquences plus propres et un dataset vidéo plus conséquent.

Table 2.2 – Comparaison entre le MLP statique et le GRU temporel du Modèle 2.

Le classifieur **MLP** reconnaît 26 lettres, mais les lettres dépendantes du mouvement ou très proches visuellement restent les plus instables. Les confusions observées sur J, Y, Z, D, P et K confirment qu'une posture isolée ne suffit pas toujours. La branche **GRU** du **Modèle 2** apporte une réponse à cette limite en traitant des séquences de **keypoints**. Elle est intégrée à l'exécution en mode dynamique : l'application déclenche la capture avec une ligne de seuil, évite les frames hors geste, puis lance régulièrement une prédiction temporelle lorsque la séquence exploitable est constituée. Cette intégration rend le mode dynamique utilisable directement dans l'interface.

2.9.3. Limites scientifiques et techniques.

La première limite concerne la richesse de la **LSF**. Une vraie traduction ne dépend pas seulement des mains. Les expressions du visage, la direction du regard, l'espace de signation et le contexte grammatical sont essentiels. Ces dimensions restent hors du périmètre traité par notre démonstrateur.

La deuxième limite concerne les données et les mesures expérimentales. Les courbes d'entraînement et la matrice de confusion du **GRU** indiquent de bonnes performances de validation, mais ces résultats restent mesurés sur un jeu contrôlé. Dans l'interface réelle, l'écart est plus marqué : les **keypoints** peuvent sauter, certaines lettres peuvent se confondre, la segmentation temporelle du geste peut être imparfaite et la prédiction peut passer brièvement d'une lettre à une autre. Pour le **GRU**, environ quarante vidéos par lettre ne suffisent probablement pas à couvrir la diversité des rythmes, amplitudes et manières de signer. Une amélioration importante consisterait à collecter les données avec des associations ou des spécialistes de la **LSF**, afin d'obtenir des gestes plus cohérents, plus représentatifs et mieux annotés. Les mesures par matrice de confusion restent utiles, mais elles doivent donc être interprétées avec prudence et confrontées systématiquement à des essais en conditions réelles.

La troisième limite concerne le temps réel. En pratique, la fluidité se situe plutôt autour de 10 à 15 images par seconde et dépend du réseau, de la disponibilité du serveur **Google Cloud**, de la charge des modèles et de la taille des images. Le filtrage par seuil de confiance, la stabilité temporelle du **MLP**, le seuil de capture du mode dynamique et le délai anti-doublon améliorent l'usage, mais introduisent aussi un délai avant l'ajout d'une lettre à l'historique reconnu. Une amélioration importante consisterait à accélérer l'**inférence** avec un **GPU** côté serveur, afin d'augmenter la marge de performance, de viser une cadence plus élevée et de mieux absorber plusieurs utilisateurs simultanés.

2.10. Perspectives.

2.10.1. Amélioration de la traduction.

À court terme, la priorité est d'augmenter et de diversifier le [dataset](#), puis de comparer plus systématiquement les architectures. La matrice de confusion permet déjà d'identifier les lettres problématiques et d'adapter le protocole de collecte. Un module de validation de confiance pourrait également éviter d'afficher des traductions trop incertaines.

À moyen terme, l'intégration d'un modèle de langage pourrait améliorer la sortie textuelle. Il ne s'agirait pas de remplacer la reconnaissance des lettres, mais de corriger ou structurer une suite de caractères lorsque le système sera suffisamment fiable. Un [LLM](#) pourrait aider à adapter la syntaxe, gérer les conjugaisons et produire un français plus naturel à partir d'une reconnaissance brute. Cette étape doit rester encadrée, car un modèle de langage peut corriger mais aussi inventer.

L'amélioration de la traduction reste nécessairement progressive. Le passage de lettres isolées à des mots puis à des phrases complètes change la nature du problème : il ne suffit plus de reconnaître une forme, il faut comprendre une structure linguistique. La [LSF](#) utilise l'espace, le regard, les expressions du visage et l'ordre des signes d'une manière différente du français écrit. Une évolution sérieuse doit donc être menée avec des personnes compétentes en [LSF](#), afin d'éviter de produire une traduction techniquement impressionnante mais linguistiquement incorrecte.

2.10.2. Synthèse vocale et accessibilité.

Une fois le texte obtenu, un module de synthèse vocale peut produire une sortie audio. Cette fonctionnalité renforcerait l'utilité du système dans des échanges rapides. Elle soulève toutefois des questions d'usage : l'utilisateur doit pouvoir contrôler ce qui est prononcé, corriger une erreur et choisir de rester en sortie texte uniquement.

Cette fonctionnalité doit être pensée avec prudence. Une traduction orale directe peut être utile dans un commerce, à un guichet ou pendant un échange court, mais elle ne doit pas priver l'utilisateur du contrôle sur son message. L'interface doit donc permettre de valider le texte avant lecture audio, ou au minimum de désactiver la synthèse vocale. L'accessibilité ne consiste pas seulement à ajouter une sortie vocale, mais bien à laisser à la personne le choix de la manière dont elle souhaite communiquer.

2.10.3. Dimension pédagogique.

La dimension pédagogique peut être prolongée au-delà de l'alphabet. Une première amélioration consisterait à ajouter progressivement des signes usuels, puis des exercices organisés par thèmes : salutations, besoins simples, démarches administratives ou vocabulaire scolaire. Le système pourrait aussi proposer des parcours plus progressifs, avec des niveaux, des rappels espacés et une adaptation automatique aux lettres ou signes les moins bien maîtrisés.

Une seconde évolution porterait sur la validation des gestes. Pour les signes dynamiques, la correction ne devrait pas seulement vérifier la classe finale, mais aussi la trajectoire, le rythme et la stabilité des [points clés](#). À plus long terme, cette partie apprentissage pourrait donc devenir un véritable outil d'entraînement à la [LSF](#), à condition d'être conçue avec des personnes compétentes en langue des signes pour éviter de réduire l'apprentissage à une simple reproduction mécanique.

2.10.4. Exploration théorique d'un modèle économique.

Cette partie constitue une réflexion prospective de long terme. Elle ne décrit pas un modèle commercial déjà prêt à être déployé avec le prototype actuel ; elle sert plutôt à vérifier que les choix techniques et

sociaux du projet pourraient, plus tard, s'inscrire dans une logique viable. Le choix d'un modèle économique responsable reste central pour la crédibilité du projet. HearMyHands n'a pas vocation à transformer un besoin d'inclusion en produit réservé aux organisations capables de payer cher. L'accès à la communication doit rester un droit et non un privilège financier. Nous envisageons donc, à titre théorique, un modèle dans lequel le secteur professionnel financerait l'infrastructure, tandis que l'accès individuel demeurerait gratuit ou très peu coûteux.

Analyse des coûts : du prototype à l'échelle industrielle. Notre budget prévisionnel de sécurité était de 900 euros. Il devait couvrir des dépenses ponctuelles en cas de blocage, par exemple l'achat de webcams, la location temporaire d'un serveur ou quelques essais de calcul pour l'entraînement. Le budget effectivement dépensé est de 0 euro. Ce résultat s'explique par l'emprunt de matériel auprès de l'école, l'utilisation de nos ordinateurs personnels, l'accès à des ressources de calcul prêtées par le LS2N et l'exploitation d'offres d'essai gratuites, notamment [Google Cloud](#) et [Oracle](#). Cette situation correspond au stade prototype. Le passage à une solution viable et scalable nécessiterait en revanche une analyse plus complète des coûts.

Pour mesurer l'écart entre notre projet étudiant et un développement professionnel complet, nous avons aussi construit une estimation théorique en partant de zéro. Elle ne correspond pas à une dépense réelle du projet, mais à ce qu'il faudrait prévoir si l'équipe devait être rémunérée, équipée et outillée pendant six mois. Nous retenons des ordres de grandeur prudents : quatre ingénieurs IA à 5 500 euros chargés par mois, six développeurs à 4 500 euros chargés par mois et deux community managers à 3 200 euros chargés par mois.

Poste théorique	Hypothèse de calcul	Coût estimé
Ingénieurs IA	4 personnes pendant 6 mois, 5 500 euros par mois	132 000 euros
Développeurs	6 personnes pendant 6 mois, 4 500 euros par mois	162 000 euros
Community managers	2 personnes pendant 6 mois, 3 200 euros par mois	38 400 euros
Postes portables	8 ordinateurs portables à 600 euros	4 800 euros
Tours avec carte graphique	4 tours à 1 500 euros	6 000 euros
Abonnements LLM	8 abonnements à 20 euros par mois pendant 6 mois	960 euros
Budget de sécurité	Webcams, serveur ponctuel et imprévus	900 euros
Total hors entraînement industriel	Développement complet équipé et rémunéré	345 060 euros

Table 2.3 – Estimation théorique d'un budget complet si le projet devait être mené sans matériel prêté ni travail étudiant bénévole.

Cette estimation peut être complétée par le coût d'entraînement à grande échelle présenté ci-dessous. En ajoutant une enveloppe de 15 000 à 30 000 euros pour un entraînement industriel sur [dataset](#) beaucoup plus large, le coût total théorique atteindrait environ 360 000 à 375 000 euros. Ce chiffre sert uniquement à donner un ordre de grandeur du passage d'un projet académique à un projet professionnalisé.

Les coûts de développement et d'entraînement correspondent au **CAPEX**. L'amélioration de la précision, notamment pour le **GRU** dynamique du **Modèle 2**, nécessiterait à long terme des [datasets](#) beaucoup plus massifs et plusieurs cycles d'apprentissage. À l'échelle d'un [dataset](#) national, l'entraînement complet d'un modèle robuste pourrait nécessiter une location de puissance de calcul haute performance, par exemple des **GPU NVIDIA A100** ou **H100** via [Google Cloud](#) ou [AWS](#). L'estimation de 15 000 à 30 000 euros doit donc être comprise comme un ordre de grandeur théorique pour réfléchir au passage à l'échelle,

et non comme un besoin immédiat du prototype. Ce coût pourrait être réduit par du [transfer learning](#) et par des partenariats cloud solidaires, comme Microsoft AI for Good ou Google for Startups.

Les coûts d'exploitation correspondent à l'[OPEX](#). Ils incluent l'[inférence](#) en ligne, l'hébergement de l'application, la maintenance des serveurs, la correction orthographique ou grammaticale par [LLM](#), la synchronisation éventuelle des profils utilisateurs, la surveillance du service et les mises à jour de sécurité. Le déploiement actuel sur [Google Cloud](#) constitue une première étape de cette logique : il permet d'évaluer l'accès distant depuis téléphone ou ordinateur, sans faire dépendre l'expérience de la puissance locale du terminal.

Stratégie de diffusion : la force de l'open source. Notre projet repose sur une philosophie d'ouverture. Inspirés par des acteurs comme Mistral AI et par des outils déjà ouverts comme [MediaPipe](#), nous voulons rendre public le cœur de la technologie : le moteur de reconnaissance de [squelette](#), les modèles de traduction et la documentation nécessaire à leur réutilisation. Cette ouverture permettrait à d'autres équipes d'améliorer les modèles, de vérifier nos choix et d'adapter l'outil à des usages que nous ne pouvons pas couvrir seuls.

L'objectif est d'offrir une base technologique complète à la communauté concernée, afin que l'outil puisse évoluer vers différents dialectes de signes, différents contextes d'usage et différents niveaux d'apprentissage sans repartir de zéro. Cette stratégie est cohérente avec la finalité sociale du projet : plus la base est ouverte, plus elle peut être corrigée, enrichie et appropriée par les utilisateurs, les associations, les chercheurs et les développeurs.

Financement B2B au service de la gratuité. La source principale de revenus envisagée à long terme serait le [B2B](#). Elle ciblerait les organisations ayant des obligations d'accessibilité ou un intérêt direct à fluidifier leurs échanges avec des personnes sourdes ou malentendantes. Une première offre pourrait prendre la forme de licences [API](#) vendues à des plateformes de visioconférence, de service client ou de relation usager. Des acteurs comme Zoom, Teams ou des plateformes de support sont cités ici comme exemples de réflexion stratégique, pas comme partenaires ou objectifs commerciaux immédiats.

Une seconde offre viserait les établissements recevant du public : mairies, hôpitaux, universités, banques ou guichets administratifs. Un abonnement professionnel pourrait proposer une interface de traduction illimitée, une assistance prioritaire, une meilleure disponibilité serveur et des garanties de maintenance. Le principe reste que les usages institutionnels financent l'amélioration et l'exploitation du service, sans rendre l'application grand public dépendante d'un abonnement élevé.

Application grand public : freemium éthique. Le mode apprentissage déjà présent dans l'application web confirme l'intérêt d'un usage pédagogique grand public. À plus long terme, une application mobile dédiée à l'apprentissage de la [LSF](#), inspirée de modèles pédagogiques comme Duolingo, devrait rester largement accessible. Les modules de cours, les exercices progressifs et la correction par webcam peuvent rester gratuits afin d'encourager la diffusion de la [LSF](#) auprès du grand public.

La monétisation individuelle doit rester légère. Une première possibilité est l'affichage de publicités solidaires courtes entre deux leçons. Une seconde possibilité est une option premium à achat unique, par exemple 4,99 euros, supprimant définitivement la publicité. Ce modèle [freemium](#) est plus cohérent avec nos valeurs qu'un abonnement mensuel : l'utilisateur contribue une fois à la pérennité du projet, sans que l'accès à l'apprentissage devienne une charge régulière.

2.11. Conclusion générale.

HearMyHands est né d'une problématique simple à formuler mais difficile à résoudre : comment faciliter la communication entre des personnes qui ne partagent pas la même langue, sans imposer un équipement coûteux ou intrusif ? Nous apportons une réponse fondée sur une interface web et une chaîne de traitement par [intelligence artificielle](#).

Nous assumons un périmètre maîtrisé. Ce choix correspond à une démarche d'ingénierie réaliste. La [LSF](#) est une langue complète, avec une grammaire propre, et une traduction automatique fiable ne peut pas être obtenue uniquement en reconnaissant quelques positions de main. En revanche, la reconnaissance de signes simples constitue une base utile pour progresser vers des usages plus ambitieux.

Au-delà du résultat technique, nous avons appris à cadrer un besoin, organiser un groupe nombreux, gérer un [dataset](#), dialoguer avec des partenaires et faire évoluer le périmètre face aux contraintes. Les prochaines étapes consistent à enrichir les données, renforcer la robustesse et associer davantage de personnes maîtrisant la [LSF](#).

3

Travaux complémentaires

3.1. Essais réalisés en parallèle.

Plusieurs essais ont été menés en parallèle de la chaîne principale. Ils ne remplacent pas les développements décrits dans le chapitre technique, mais ils ont permis de sécuriser l'intégration progressive du projet et de limiter les risques avant l'assemblage complet.

- Tests isolés des briques avant leur intégration dans l'application, afin de vérifier leur comportement sans dépendre immédiatement de toute la chaîne.
- Vérification des formats d'échange entre les outils, notamment pour éviter les incohérences entre vidéos, [JSON](#), annotations et [keypoints](#).
- Essais de robustesse en conditions d'usage, avec des cadrages, éclairages et positions de main différents de ceux utilisés pendant la mise au point.
- Préparation de supports visuels et de captures intermédiaires pour analyser les erreurs, expliquer les choix techniques et faciliter la maintenance.

Les sections suivantes détaillent uniquement les travaux complémentaires les plus significatifs : la recherche d'hyperparamètres du [MLP](#), les essais de loss pour le [Modèle 1](#), l'outil de segmentation vidéo avec export [COCO](#), puis l'exploration [Blender](#).

3.2. Recherche d'hyperparamètres du MLP.

Le script `training_optimized.py` a servi à rendre l'entraînement du [MLP](#) statique plus rapide et plus reproductible. L'objectif n'était pas seulement d'obtenir un bon score, mais aussi de tester proprement plusieurs choix d'[hyperparamètres](#) sur un jeu d'évaluation séparé. Le script charge donc deux ensembles distincts : `training_keypoints.json` et `training_label.json` pour l'apprentissage, puis `eval_keypoints.json` et `eval_label.json` pour l'évaluation. Cette séparation évite de sélectionner un modèle uniquement parce qu'il a mémorisé les exemples d'entraînement.

La recherche porte sur quatre paramètres principaux. Le taux d'apprentissage contrôle l'amplitude des corrections appliquées aux poids du réseau ; nous avons testé plusieurs valeurs, notamment 0,1, 0,5 et 0,9. La taille de la couche cachée fixe la capacité du modèle à combiner les 42 coordonnées d'entrée ; la configuration conservée utilise 50 neurones cachés. Le nombre d'itérations détermine la

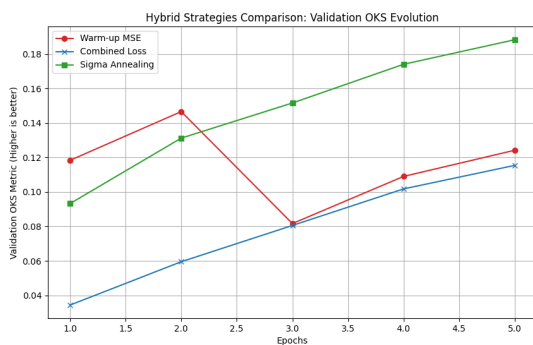
durée de l'apprentissage, avec 50 000 passages d'optimisation pour les essais retenus. Enfin, la fonction d'activation choisie est **ReLU**, car elle donne une non-linéarité simple et rapide à calculer.

Le réseau testé dans ce script conserve une sortie softmax et une loss **MSE**, puis il est optimisé avec SGD. À chaque configuration, le modèle est entraîné, évalué sur le jeu séparé, puis ses résultats sont enregistrés dans `hyperparameters_results.json`. Le script produit aussi des statistiques détaillées par lettre, avec vrais positifs, faux positifs et faux négatifs. Cette étape a permis de retenir la configuration $42 \rightarrow 50 \rightarrow 26$, avec un taux d'apprentissage de 0,9, une activation ReLU et 50 000 itérations. Les poids finaux sont ensuite exportés dans quatre fichiers **JSON**, `W1.json`, `W2.json`, `b1.json` et `b2.json`, afin que l'application puisse exécuter le MLP en production avec NumPy plutôt qu'avec PyTorch.

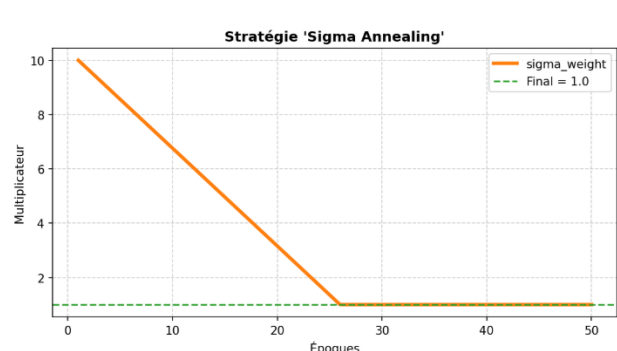
3.3. Choix de la loss du Modèle 1.

Un point important du **Modèle 1** a été le choix de la loss utilisée pour entraîner les points clés du haut du corps. Une loss **MSE** directe sur les coordonnées apprend rapidement au début, mais elle ne tient pas compte de l'échelle du corps ni de la tolérance différente selon les articulations. À l'inverse, l'**OKS** est plus cohérent pour la pose humaine, mais son gradient peut être faible au début de l'entraînement lorsque les prédictions sont encore loin de la vérité terrain.

Nous avons donc comparé trois stratégies : utiliser uniquement l'**OKS**, commencer par une phase en **MSE** puis basculer vers l'**OKS**, ou utiliser un *OKS annealing*. Cette dernière stratégie est celle retenue. Elle consiste à renforcer l'effet de la loss au début par un multiplicateur élevé, puis à diminuer progressivement ce facteur jusqu'à revenir à une pondération de 1. La figure 3.1a compare les comportements observés entre les loss testées, tandis que la figure 3.1b montre la décroissance du multiplicateur utilisé pour l'annealing. Le modèle bénéficie ainsi d'un signal plus fort dans les premières époques, tout en conservant à la fin le comportement normal de l'**OKS**.



(a) Comparaison MSE/OKS



(b) Sigma annealing

Figure 3.1 – Essais de loss pour le Modèle 1 : la comparaison MSE/OKS aide à identifier les limites des loss testées, et le sigma annealing montre le multiplicateur élevé au début de l'entraînement puis ramené progressivement à 1.

3.4. Segmentation vidéo et annotations COCO.

Pour accélérer la constitution du **dataset** temporel destiné au **GRU**, nous avons développé un outil de segmentation vidéo, principalement porté par Marius. Il s'agit d'une application **Flask** indépendante du démonstrateur principal. Son rôle est de transformer une vidéo longue de **LSF** en clips courts exploitables pour l'entraînement des lettres dynamiques et l'analyse des trajectoires.

Le workflow est structuré en plusieurs étapes. Nous importons d'abord une vidéo dans l'interface. L'application lance ensuite un scan automatique avec **MediaPipe Hands** en mode vidéo. Ce mode utilise les timestamps des frames et permet de suivre les mains dans le temps au lieu de traiter chaque image comme une photographie isolée. Les zones où la main active dépasse un seuil vertical sont proposées comme segments. Nous pouvons ensuite corriger manuellement les débuts, fins et labels des segments avant export.

L'export produit des clips courts au format vidéo. La fonction de découpage applique un **padding** avant et après le segment afin d'éviter de supprimer le début ou la fin du geste. Les noms de fichiers sont normalisés à partir des labels, ce qui rend les données directement exploitables par les scripts suivants. Cet outil concerne donc surtout le **GRU** : le **MLP** statique dépend principalement d'images ou de **keypoints** isolés, alors que le modèle temporel a besoin de séquences homogènes.

L'outil de segmentation n'est pas limité à une utilisation strictement locale. Il a également été rendu accessible à l'adresse <http://hnh-segmentation.duckdns.org/>. Une redirection réseau via Nginx permet de travailler à plusieurs sur la même machine de traitement, sans avoir à dupliquer l'environnement de segmentation sur chaque ordinateur.

Après le découpage, la chaîne d'annotation extrait les 21 points de la main **MediaPipe** pour chaque clip et écrit un fichier **JSON** par vidéo au format **COCO** adapté. Chaque annotation contient notamment l'identifiant de frame, la boîte englobante, le nombre de points, les coordonnées **x**, **y**, la visibilité, le label **Left/Right**, le label brut **MediaPipe** et un **track_id**. Cette structure, illustrée en annexe A, rend possible un contrôle image par image et facilite l'entraînement de modèles qui attendent une représentation normalisée des articulations.

Nous avons ajouté plusieurs mécanismes pour fiabiliser les annotations. Le paramètre **-hand** permet de filtrer la main gauche, la main droite ou les deux. Le paramètre **-side** filtre la position à l'écran afin de standardiser les vidéos. Le paramètre **-handedness-mode swap** corrige les cas où une vidéo non miroir conduit à une inversion gauche/droite. Enfin, les options d'interpolation et de lissage comblent les petites pertes de détection et réduisent les sauts ponctuels de **keypoints**.

La qualité d'un **dataset** de signes dépend fortement de la qualité des annotations. Nous avons donc prévu des contrôles visuels permettant de rendre un fichier **COCO** sous forme de vidéo **squelette** sur fond noir et de vérifier rapidement les segments exportés. Deux opérations complètent cette chaîne de nettoyage : l'inversion des labels **Left** et **Right** lorsque la convention de main n'est pas cohérente, et le miroir horizontal des coordonnées lorsque cela permet d'homogénéiser les données.

Le protocole du **dataset** temporel repose sur plusieurs prises par personne. Pour chaque signe, chaque participant réalise deux prises, ce qui permet d'obtenir environ quarante exemples par signe lorsque le nombre de participants est suffisant. Les clips obtenus sont ensuite complétés par l'augmentation de données et intégrés au pipeline d'entraînement du **GRU**.

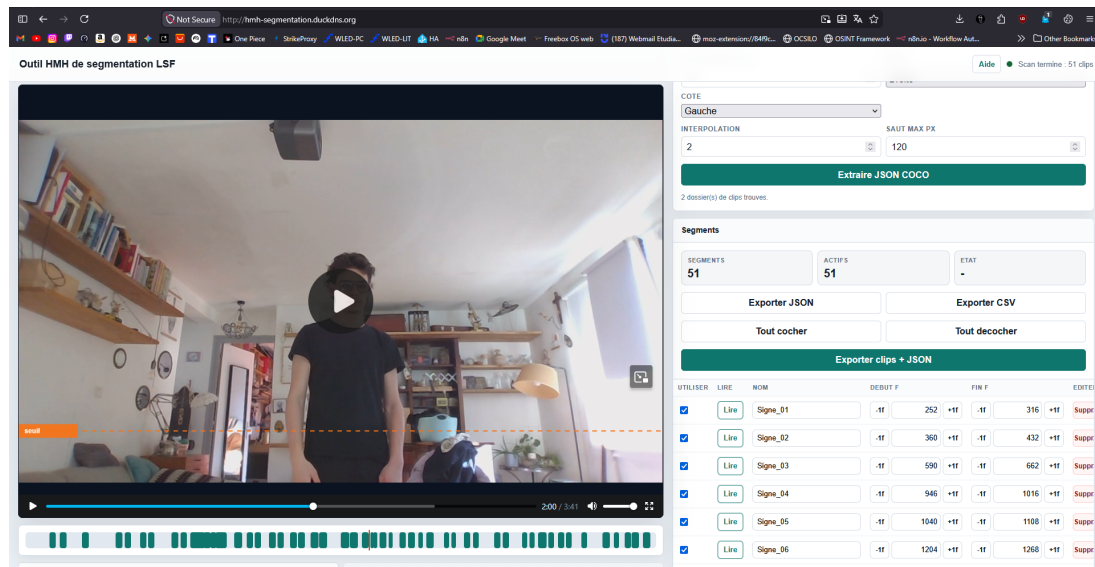


Figure 3.2 – Interface de segmentation vidéo utilisée pour préparer des clips courts destinés au GRU.

3.5. Exploration Blender.

En parallèle des données réelles, nous avons exploré une génération synthétique avec **Blender**. L'objectif est de produire des images annotées automatiquement, sans annotation manuelle, afin d'augmenter la diversité des positions de bras, de mains, d'éclairages et de points de vue.

La chaîne de génération orchestre des rendus **Blender** en arrière-plan, crée des poses aléatoires centrées sur les mains et les avant-bras, puis exporte une image et une annotation associée. À chaque rendu, elle peut faire varier la pose, l'arrière-plan, la couleur du monde, les lumières et la position de la caméra. Un outil de visualisation superpose ensuite le **squelette** sur les rendus récents afin de vérifier que les points projetés correspondent bien à l'image générée.

Cette piste a surtout servi d'exploration au début du projet, au moment où nous cherchions une manière d'obtenir rapidement un **dataset** annoté. Elle s'est révélée intéressante pour comprendre la génération contrôlée de données, mais trop complexe pour produire seule un **dataset** réellement généralisable. Les écarts entre poses synthétiques, rendu 3D et gestes humains réels risquaient d'introduire un décalage entre l'entraînement et l'usage final. Nous avons donc choisi de privilégier les **datasets** réels avec de vraies personnes, tout en conservant **Blender** comme piste complémentaire et comme outil de réflexion sur l'augmentation de données.

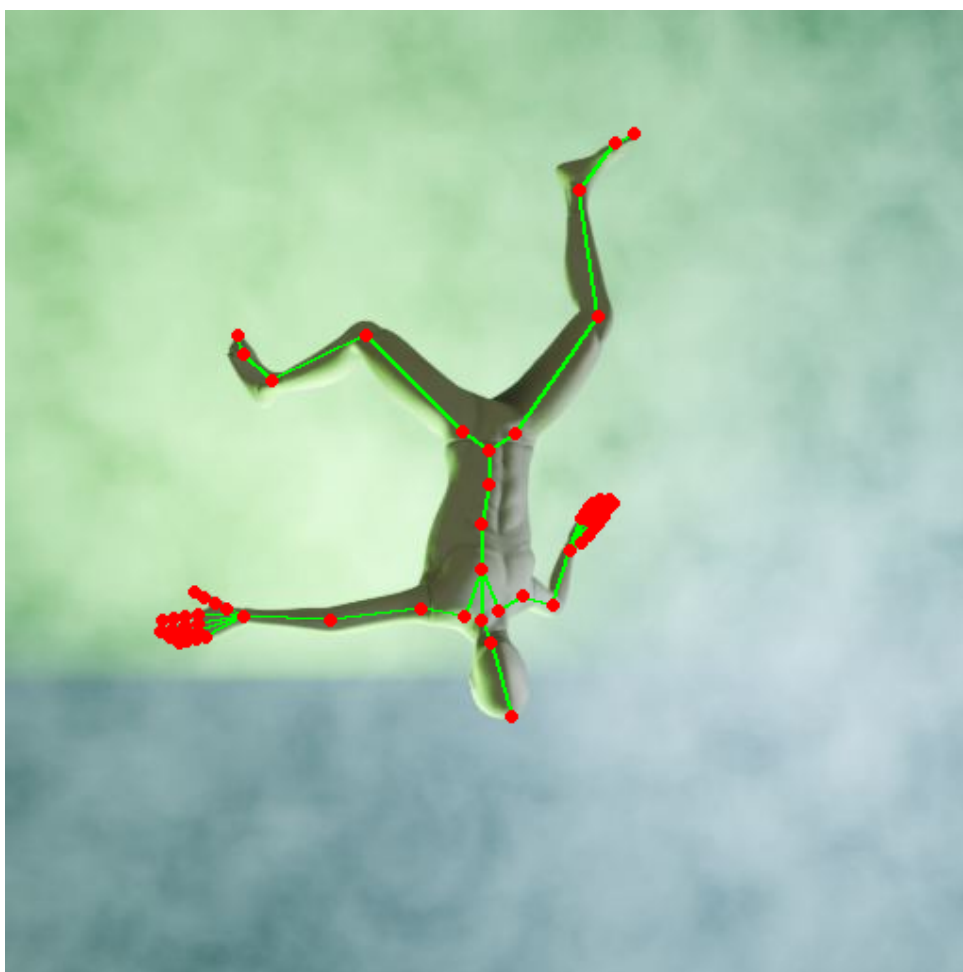


Figure 3.3 – Exemple d'image synthétique Blender avec superposition des points du squelette.

4

Aspects gestion de projet

4.1. Organisation de l'équipe.

Notre équipe est composée de douze membres et a été coordonnée par Marius, chef de projet. Au tout début du projet, nous avons diffusé un questionnaire interne au sein du groupe afin d'identifier les compétences, les préférences et les envies de chacun. Cette étape nous a aidés à répartir les tâches de manière plus cohérente : les membres les plus intéressés par la vision et l'apprentissage automatique se sont orientés vers les modèles, tandis que d'autres ont davantage contribué à l'interface, à la communication, à la collecte de données ou à l'organisation.

Pour rendre le projet gérable, nous avons ensuite réparti le travail en plusieurs pôles principaux. Le pôle **Modèle 1**, composé notamment de Maxime, Thibault, Titouan et François, s'est concentré sur la vision : détection du haut du corps, intégration de **MediaPipe Hands** et production du **squelette** exploitable. Le pôle **Modèle 2**, composé de Killian, Dalyan, Nathan, Louna et Marius, a pris en charge la traduction, avec le **MLP** pour les lettres statiques et le **GRU** pour les lettres dynamiques. Le pôle communication, composé de Louna, Arthur et Nestor, a géré les échanges avec les professeurs et les associations, mais a aussi contribué à la constitution du **dataset**, notamment pour le **GRU**. Cette répartition nous a permis d'avancer en parallèle, mais elle a aussi rendu nécessaire une coordination régulière pour éviter les incompatibilités entre formats de données.

Certaines responsabilités ont été plus transversales. Kaelig a principalement travaillé sur l'interface, l'intégration et la direction artistique de l'application. Maxime a également contribué à la gestion de l'**API** et l'intégration des différents modules. La préparation du trailer et des supports de communication a surtout mobilisé Kaelig, Killian et François. Cette organisation reflète la réalité du projet : les pôles structurent le travail, mais plusieurs livrables nécessitent des contributions croisées.

Le fait de travailler dans une équipe nombreuse a été à la fois une force et une contrainte. Il nous a permis de traiter plusieurs sujets en parallèle, de partager la charge d'apprentissage et de maintenir une dynamique collective sur un projet long. En contrepartie, il a rendu plus difficile le suivi individuel des tâches, la synchronisation des décisions techniques et la conservation d'une vision commune. Notre gestion de projet a donc consisté moins à répartir simplement le travail qu'à maintenir une cohérence entre des contributions très différentes.

Chaque pôle a produit des livrables compréhensibles par les autres. Par exemple, le pôle communication et **dataset** fournit des fichiers nommés et structurés de manière compatible avec l'entraînement ; le pôle **Modèle 1** précise les points disponibles en sortie du module de vision ; le pôle **Modèle 2** définit le format

attendu par les classifieurs ; l'interface anticipe le format des résultats renvoyés par le backend. Cette dépendance entre sous-groupes impose une communication régulière.

La coordination repose donc sur une alternance entre autonomie et synchronisation. Chaque sous-groupe peut avancer sans attendre une validation permanente, mais les décisions structurantes sont partagées rapidement. Cette organisation est particulièrement importante lorsque certains choix techniques ont des conséquences sur plusieurs pôles, par exemple le format des données extraites ou la manière de découper les vidéos du [dataset](#).

4.2. Outils utilisés.

Nous avons organisé les échanges quotidiens via Discord, avec des canaux dédiés aux différents sous-sujets. Nous avons mis en place un bot pour rediriger automatiquement les courriels de l'adresse universitaire collective vers les salons de discussion, ce qui limite la perte d'information. Nous versionnons le code sur GitLab et GitHub afin de conserver l'historique des modifications et de faciliter le travail à plusieurs. Nous utilisons ClickUp pour suivre la planification via un diagramme de Gantt interactif.

Ces outils répondent à trois besoins différents. Discord sert à la communication rapide et à la résolution de blocages ponctuels. GitLab et GitHub conservent une trace du code et évitent la multiplication de versions locales impossibles à réconcilier. ClickUp permet de visualiser le planning global, les dépendances et les retards. L'enjeu principal n'est pas d'utiliser beaucoup d'outils, mais de donner à chacun une fonction claire.

L'utilisation d'outils collaboratifs permet également de conserver une mémoire du projet. Dans un projet long, les décisions prises au début peuvent être oubliées ou remises en question plusieurs semaines plus tard. Les salons thématiques, les dépôts de code et le planning servent de points de référence pour retrouver les choix effectués, les tâches attribuées et les problèmes déjà identifiés.

Nous avons aussi encadré l'usage des [LLM](#), notamment ChatGPT, Gemini et Claude. Ces outils ont été utilisés comme aides pour comprendre une documentation, comparer des pistes, expliquer une erreur ou accélérer un débogage. Nous avons volontairement alterné entre plusieurs modèles afin d'obtenir des formulations différentes et de limiter le risque de suivre une seule réponse sans recul. Les informations importantes sont vérifiées manuellement, et le code proposé par un [LLM](#) n'est intégré que s'il est relu, compris, adapté au projet et assumé par nous. Cette règle permet de bénéficier de ces outils sans perdre la maîtrise de l'architecture, des choix techniques et de la traçabilité du travail.

4.3. Difficultés rencontrées.

La première difficulté est scientifique. Les notions de vision par ordinateur, de réseaux de neurones et de traitement séquentiel demandent un temps d'appropriation important. Nous avons appris à manipuler des bibliothèques comme [MediaPipe](#), OpenCV et [PyTorch](#) tout en comprenant les limites de chaque outil.

La deuxième difficulté est liée aux données. Concevoir un modèle est relativement rapide ; obtenir un [dataset](#) propre, homogène, annoté et assez diversifié est beaucoup plus long. La qualité des vidéos influence directement les performances, ce qui nous a imposé de formaliser un protocole précis.

La troisième difficulté est organisationnelle. Un groupe de douze personnes permet de répartir le travail, mais augmente les besoins de coordination. Nous avons clarifié les formats de fichiers, les conventions de nommage et les interfaces entre modules pour éviter que chaque sous-groupe développe une solution isolée.

Un autre choix organisationnel important est le recours majoritaire au distanciel. Cette organisation réduit les temps de trajet, mais elle est aussi particulièrement adaptée à un projet informatique : partage d'écran, accès aux dépôts Git, tests de code, rédaction collaborative et résolution de bugs peuvent être faits efficacement à distance. Elle peut toutefois diminuer la motivation et l'implication si elle n'est pas

structurée. Nous avons donc ajusté progressivement notre méthode de suivi en clarifiant les tâches à traiter, les points de blocage et les livrables attendus avant chaque échange.

La gestion humaine est également un point d'apprentissage. Tous les membres ne progressent pas au même rythme et tous n'ont pas la même aisance initiale en programmation. Nous avons donc adopté une répartition évolutive des rôles : certains membres prennent davantage de responsabilités techniques, tandis que d'autres contribuent sur la collecte, la vérification des données, l'organisation ou la communication. Cette adaptation préserve l'avancement global sans réduire la valeur des contributions non techniques.

4.4. Analyse des risques.

Risque	Impact	Réponse mise en place
Dataset insuffisant	Modèle peu généralisable	Protocole de captation et sollicitation de personnes externes.
Erreur de keypoints	Mauvaise classification	Normalisation, contrôle visuel et possibilité d'ajouter un seuil de confiance.
Retard technique	Démonstration incomplète	Réduction du périmètre aux lettres de l'alphabet.
Désalignement des sous-groupes	Modules incompatibles	Conventions de format et réunions de synchronisation.
Baisse de motivation	Rythme irrégulier	Découpage en tâches courtes et visibles.
Distanciel majoritaire	Isolement et perte d'attention	Canaux Discord mieux structurés et suivi plus explicite des tâches.

Table 4.1 – Principaux risques du projet et réponses associées.

Cette analyse des risques évolue au fil du projet. Au départ, les risques les plus visibles semblaient surtout techniques : réussir à entraîner un modèle, obtenir une reconnaissance correcte, ou faire fonctionner une interface. Avec l'avancement, d'autres risques apparaissent, notamment la qualité du **dataset**, la motivation en distanciel et la difficulté à maintenir une cohérence entre sous-groupes. Cette évolution montre que la réussite du projet ne dépend pas uniquement du code, mais aussi de notre capacité à anticiper et à corriger notre organisation.

4.5. Évolution du planning.

Le planning initial n'était pas irréaliste sur le papier, mais nous avons eu du mal à le respecter. Collecte de données, entraînement, interface et démonstration complète dépendaient les uns des autres, et certains blocages ont retardé l'ensemble de la chaîne. Au fil du projet, nous avons ajusté le périmètre pour privilégier une reconnaissance fiable des lettres de l'alphabet plutôt qu'une accumulation de fonctionnalités inachevées. Ce choix est cohérent avec une démarche d'ingénierie : un système maîtrisé apporte plus de valeur qu'un système très large mais instable.

L'évolution du planning montre aussi que le projet ne suit pas une progression linéaire. Certaines tâches, comme la compréhension des modèles ou la mise au point du [dataset](#), prennent plus de temps que prévu car elles conditionnent toutes les étapes suivantes. Nous avons aussi connu une phase de démarrage trop lente sur certaines parties : plusieurs décisions ont été repoussées, notamment sur le périmètre exact, les formats de données. Cette inertie initiale a concentré une grande partie de la production effective sur les deux derniers mois du projet. Ce constat fait partie du bilan : il montre l'importance de transformer plus tôt les intentions en tâches concrètes, datées et vérifiables.

Le diagramme de Gantt initial reflète cette volonté de structurer le travail sur la durée, mais il ne correspond pas parfaitement au rythme réel d'exécution. Il a surtout servi de support de projection et de discussion. Avec l'avancement, nous avons privilégié une planification plus courte, centrée sur les blocages immédiats et les livrables nécessaires à la démonstration finale.

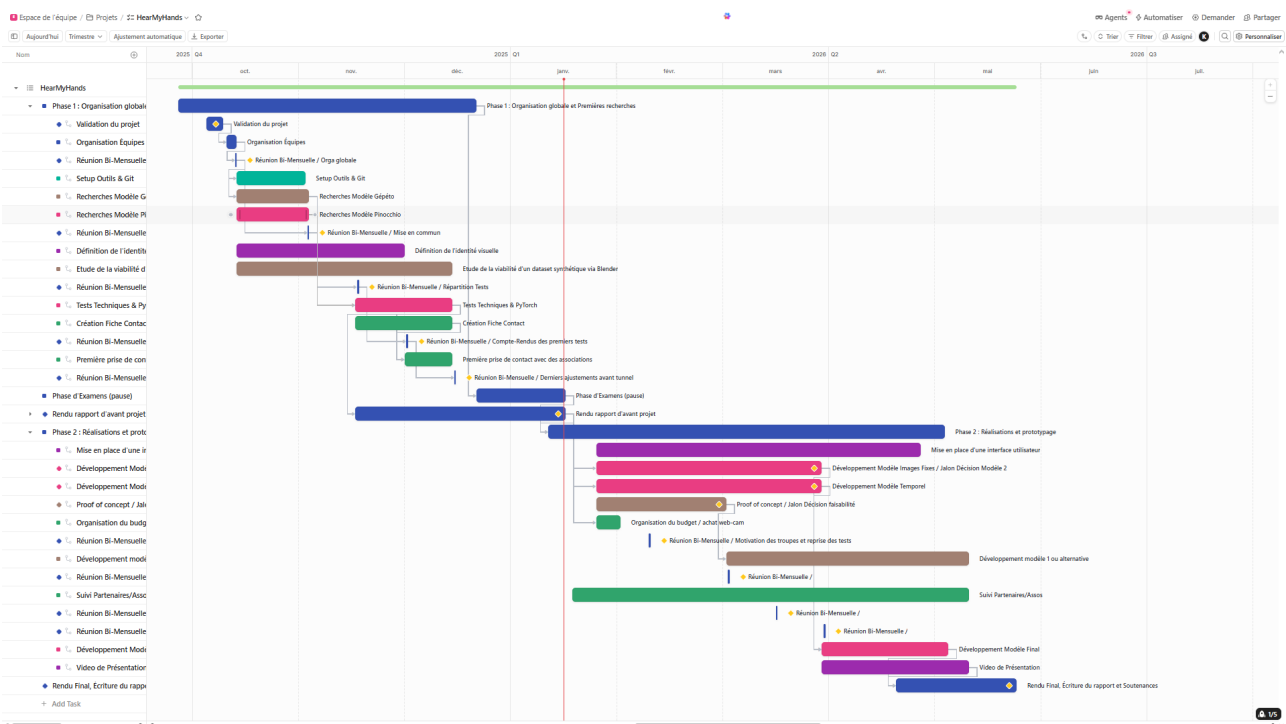


Figure 4.1 – Diagramme de Gantt initial utilisé pour planifier les grandes étapes du projet.

À l'inverse, nous mettons de côté certaines idées initiales lorsqu'elles risquent de disperser l'équipe. Cette priorisation progressive permet de conserver un objectif clair : présenter une solution cohérente, démontrable et techniquement maîtrisée.

4.6. Bilan de gestion.

La gestion du projet HearMyHands repose donc sur trois principes : découper le problème, rendre les échanges visibles et accepter d'adapter le périmètre. Ces choix permettent à notre équipe nombreuse et hétérogène de produire un travail cohérent malgré les difficultés techniques. Notre principale amélioration identifiée concerne la formalisation plus précoce des interfaces entre sous-groupes, notamment les formats exacts des données échangées entre la collecte, les modèles et l'interface.

Ce bilan est globalement positif car nous avons su transformer un sujet très large en projet structuré. Nous avons ajusté les objectifs initiaux sans abandonner l'ambition principale : proposer un outil d'inclusion fondé sur l'[intelligence artificielle](#). La démarche suivie montre une progression importante, autant sur les compétences techniques que sur les compétences de gestion : planifier, répartir, communiquer, décider et documenter.

Remerciements

Nous remercions Hélène Pérennou, notre tutrice pédagogique, pour son suivi et ses retours tout au long du projet. Nous remercions également Matthieu Perreira Da Silva pour son accompagnement technique et ses conseils, notamment lors des phases de cadrage, de modélisation et de mise en oeuvre.

Nous remercions aussi Coralie Castel et Je m'InSigne pour leurs réponses au questionnaire et pour leur contribution à notre compréhension des usages réels de la [LSF](#). Nous remercions enfin les personnes ayant répondu depuis les groupes Facebook et forums *Sourds Oralistes*, *Forum AFIDEO*, *les sourds a la France*, *France sourds*, *Les SOURDS de la DIVERSITÉ*, *Tous Ensemble !* et *Sourds et Malentendants*. Leurs remarques ont aidé à cadrer les limites et les objectifs du projet.

5

Bibliographie

- [1] FNSF. Fédération nationale des sourds de France. <https://www.fnsf.org/>, s. d.
- [2] Nations Unies, ODD 10. Objectif de développement durable 10 : Inégalités réduites. <https://www.un.org/sustainabledevelopment/fr/inequality/>, s. d.
- [3] Nations Unies, ODD 4. Objectif de développement durable 4 : Éducation de qualité. <https://www.un.org/sustainabledevelopment/fr/education/>, s. d.
- [4] Fédération Nationale des Sourds de France. Prise de position : Interprétation et traduction. https://www.fnsf.org/wp-content/uploads/2025/05/FNSF_Prise-de-position_Interpretation-et-traduction.pdf, 2025.
- [5] Nations Unies. Convention internationale relative aux droits des personnes handicapées. <https://www.un.org/esa/socdev/enabled/documents/tccconvf.pdf>, 2006.
- [6] Fédération Nationale des Sourds de France. Prise de position : Éducation. https://www.fnsf.org/wp-content/uploads/2025/05/FNSF_Prise-de-position_Education.pdf, 2025.
- [7] Coralie Castel. Chaîne et contenus de sensibilisation autour de la langue des signes. Chaîne YouTube et réponse au questionnaire HearMyHands, s. d. <https://www.ladepeche.fr/2024/09/18/coralie-castel-une-interface-pour-la-langue-des-signes-12204654.php>.
- [8] Je m'IniSigne. Contenus pédagogiques autour de la langue des signes. Chaîne YouTube et réponse au questionnaire HearMyHands, s. d. <https://www.jeminisigne.com/a-propos-de-je-m-inisigne-a-lyon>.
- [9] Groupe Sourds Oralistes. Sourds oralistes. Groupe Facebook mobilisé pour la diffusion du questionnaire HearMyHands, s. d. <https://www.afideo.org/>.
- [10] AFIDEO. Forum afideo. Forum et communauté autour des personnes sourdes oralistes, s. d. <https://www.afideo.org/>.
- [11] Communautés Facebook LSF. Groupes facebook francophones autour de la surdité et de la langue des signes. Groupes *les sourds a la France*, *France sourds*, *Les SOURDS de la DIVERSITÉ*, *Tous Ensemble !* et *Sourds et Malentendants*, mobilisés pour la diffusion du questionnaire HearMyHands, s. d. <https://www.facebook.com/groups/>.
- [12] University of Washington News. UW undergraduate team wins lemelson-MIT student prize for gloves that translate sign language. <https://www.washington.edu/news/2016/04/12/>

- uw-undergraduate-team-wins-10000-lemelson-mit-student-prize-for-gloves-that-translate-sign-2016.
- [13] Dongxu Li, Cristian Rodriguez, Xin Yu, and Hongdong Li. Word-level deep sign language recognition from video : A new large-scale dataset and methods comparison. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2020. <https://arxiv.org/abs/1910.11006>.
 - [14] Amanda Duarte, Shruti Palaskar, Lucas Ventura, Deepti Ghadiyaram, Kenneth DeHaan, Florian Metze, Jordi Torres, and Xavier Giro-i Nieto. How2sign : A large-scale multimodal dataset for continuous american sign language. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021. <https://how2sign.github.io/>.
 - [15] RWTH Aachen University. RWTH-PHOENIX-Weather 2014 T : Parallel corpus of sign language video, gloss and translation. <https://www-i6.informatik.rwth-aachen.de/~koller/RWTH-PHOENIX-2014-T/>, s. d.
 - [16] Necati Cihan Camgoz, Simon Hadfield, Oscar Koller, Hermann Ney, and Richard Bowden. Neural sign language translation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018. https://openaccess.thecvf.com/content_cvpr_2018/html/Camgoz_Neural_Sign_Language_CVPR_2018_paper.html.
 - [17] Yanis Ouakrim, Hannah Bull, Michèle Gouiffès, Denis Beateemps, Thomas Hueber, and Annelies Braffort. Mediapi-RGB : Enabling technological breakthroughs in french sign language (LSF) research through an extensive video-text corpus. In *Proceedings of the 19th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 2 : VISAPP*, pages 139–148. SciTePress, 2024. <https://doi.org/10.5220/0012372600003660>.
 - [18] Julie Halbout, Diandra Fabre, Yanis Ouakrim, Julie Lascar, Annelies Braffort, Michèle Gouiffès, and Denis Beateemps. Matignon-LSF : a large corpus of interpreted french sign language. In *Proceedings of the LREC-COLING 2024 11th Workshop on the Representation and Processing of Sign Languages*, pages 202–208. ELRA Language Resources Association, 2024. <https://www.sign-lang.uni-hamburg.de/lrec/pub/24024.html>.
 - [19] Necati Cihan Camgoz, Oscar Koller, Simon Hadfield, and Richard Bowden. Sign language transformers : Joint end-to-end sign language recognition and translation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10023–10033, 2020. https://openaccess.thecvf.com/content_CVPR_2020/html/Camgoz_Sign_Language_Transformers_Joint_End-to-End_Sign_Language_Recognition_and_Translation_CVPR_2020_paper.html.
 - [20] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, Wan-Teh Chang, Wei Hua, Manfred Georg, and Matthias Grundmann. Mediapipe : A framework for building perception pipelines. In *Workshop on Computer Vision for Augmented and Virtual Reality, CVPR*, 2019. <https://arxiv.org/abs/1906.08172>.
 - [21] Google for Developers. Mediapipe hands. https://developers.google.com/mediapipe/solutions/vision/hand_landmarker, s. d.
 - [22] YouTube, MLP Explained. Multilayer perceptron (MLP) explained. <https://www.youtube.com/watch?v=AZEfmoWBXwg>, s. d.
 - [23] YouTube, bases mathématiques du MLP. Support vidéo sur les bases mathématiques des réseaux de neurones. <https://www.youtube.com/watch?v=bzC5cdxZc0M>, s. d.
 - [24] Valentin Bazarevsky, Ivan Grishchenko, Karthik Raveendran, Tyler Zhu, Fan Zhang, and Matthias Grundmann. BlazePose : On-device real-time body pose tracking. *arXiv preprint arXiv :2006.10204*, 2020. <https://arxiv.org/abs/2006.10204>.

- [25] Adrian Bulat and Georgios Tzimiropoulos. Human pose estimation via convolutional part heatmap regression. In *European Conference on Computer Vision*, pages 717–732. Springer, 2016. <https://arxiv.org/abs/1609.01743>.
- [26] Muhammed Can Keles, Bedrettin Cetinkaya, Sinan Kalkan, and Emre Akbas. RSPose : Ranking based losses for human pose estimation. *arXiv preprint arXiv :2511.13857*, 2025. <https://arxiv.org/abs/2511.13857>.
- [27] Xiao Sun, Bin Xiao, Fangyin Wei, Shuang Liang, and Yichen Wei. Integral human pose regression. *arXiv preprint arXiv :1711.08229*, 2017. <https://arxiv.org/abs/1711.08229>.
- [28] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8) :1735–1780, 1997. <https://doi.org/10.1162/neco.1997.9.8.1735>.
- [29] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv :1406.1078*, 2014. <https://arxiv.org/abs/1406.1078>.
- [30] YouTube Tutorial. Illustrated guide to lstm’s and gru’s. <https://www.youtube.com/watch?v=AsNTP8Kwu80>, s. d.
- [31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch : An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019. <https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html>.

6

Glossaire

- Adam** Optimiseur très utilisé en apprentissage profond. Il ajuste automatiquement les poids du modèle pendant l'entraînement à partir des gradients calculés.
- API** Interface permettant à deux logiciels de communiquer selon un format défini. Dans HearMyHands, l'API modèle reçoit les images et renvoie les points détectés ainsi que la prédiction.
- ASL** *American Sign Language*, langue des signes américaine. Elle est souvent utilisée dans les datasets et travaux de recherche internationaux, mais elle est différente de la LSF.
- AWS** Plateforme cloud d'Amazon pouvant fournir des serveurs, du stockage et des GPU pour l'entraînement ou l'inférence.
- Backbone** Partie principale d'un réseau de neurones qui extrait les caractéristiques profondes d'une image avant les couches spécialisées.
- Batch** Petit groupe d'exemples traité en même temps pendant l'entraînement d'un modèle. Utiliser des batchs stabilise et accélère l'apprentissage.
- BlazePose** Modèle de pose de MediaPipe capable d'estimer des repères corporels en temps réel, utilisé ici pour générer des annotations initiales.
- Blender** Logiciel 3D utilisé ici pour explorer la génération synthétique d'images annotées.
- B2B** Modèle économique dans lequel le service est vendu à des organisations professionnelles plutôt qu'à des particuliers.
- CAPEX** Dépenses d'investissement, par exemple l'entraînement initial d'un modèle sur une infrastructure GPU.
- Cartes de caractéristiques compactes** Représentations internes produites par un réseau de vision. Elles gardent les informations importantes de l'image, comme les formes et positions utiles, avec une résolution plus faible que l'image d'origine.
- HSV / chroma keying** Méthode de détourage fondée sur la couleur, utilisée ici pour isoler une personne filmée sur fond vert.
- Client-serveur** Architecture où le terminal utilisateur capture et affiche les données, tandis qu'un serveur exécute les traitements principaux.
- COCO** Format d'annotation couramment utilisé en vision par ordinateur pour décrire des images, des boîtes englobantes et des points clés.

- CNN** Réseau de neurones convolutionnel, très utilisé en vision par ordinateur pour extraire des caractéristiques à partir d'images.
- CSPDarknet53** Architecture convolutionnelle issue de la famille YOLO, explorée dans une version antérieure du Modèle 1 pour extraire des cartes de caractéristiques.
- Dataset** Ensemble de données utilisé pour entraîner, valider ou tester un modèle.
- Deep Learning** Famille de méthodes d'apprentissage automatique fondées sur des réseaux de neurones profonds.
- Dropout** Technique de régularisation qui désactive aléatoirement certains neurones pendant l'entraînement pour limiter le surapprentissage.
- Époque** Passage complet sur l'ensemble d'entraînement. Un entraînement de 500 époques parcourt donc 500 fois les données d'entraînement.
- Flask** Framework Python léger utilisé pour créer l'application web et l'API modèle.
- FPS** Images par seconde. Cette mesure indique la cadence du flux vidéo traité par le système.
- Freemium** Modèle économique où l'accès de base est gratuit et certaines options sont payantes.
- Google Cloud** Plateforme cloud de Google utilisée pour héberger la version en ligne de l'application et tester l'accès distant au service.
- GPU** Processeur graphique utilisé pour accélérer l'entraînement et l'inférence des modèles d'apprentissage profond.
- GRU** *Gated Recurrent Unit*. Réseau récurrent adapté au traitement de séquences, utilisé pour les lettres dynamiques.
- Gunicorn** Serveur WSGI Python utilisé pour exécuter l'application web en production, contrairement au serveur de développement lancé directement par Flask.
- Heatmap** Carte indiquant la probabilité de présence d'un point clé à chaque position de l'image.
- HTML** Langage de balisage utilisé pour structurer les pages web affichées par le navigateur.
- Hyperparamètres** Réglages choisis avant l'entraînement d'un modèle, comme le taux d'apprentissage, la taille des batchs ou le nombre d'époques. Ils influencent l'apprentissage mais ne sont pas appris automatiquement comme les poids du réseau.
- Inférence** Phase où un modèle déjà entraîné est utilisé pour prédire un résultat sur une nouvelle donnée. Dans HearMyHands, cela correspond par exemple au moment où le serveur reçoit une image de la webcam et renvoie les points détectés ou la lettre reconnue.
- Integral Pose Regression** Méthode qui convertit une heatmap en coordonnées par calcul d'espérance, ce qui rend la sortie différentiable.
- JavaScript** Langage exécuté dans le navigateur pour gérer l'interaction, la webcam et l'envoi des images vers le serveur.
- JPEG** Format d'image compressé. Un réglage de qualité plus faible réduit la taille transmise, mais peut dégrader certains détails visuels.
- JSON** Format textuel structuré utilisé pour stocker les annotations, les poids du MLP et les réponses de l'API.
- Keypoint** Point clé décrivant une articulation ou un repère du corps, par exemple un poignet, un coude ou l'un des 21 points de la main fournis par MediaPipe.
- LLM** Grand modèle de langage pouvant reformuler ou corriger une sortie textuelle.
- LSF** Langue des Signes Française, langue visuelle possédant sa propre grammaire et sa propre syntaxe.

LSTM *Long Short-Term Memory*. Réseau récurrent conçu pour traiter des dépendances temporelles longues.

Logits Scores bruts produits par un modèle de classification avant transformation en probabilités.

MediaPipe Bibliothèque de vision par ordinateur utilisée ici pour détecter les mains et leurs articulations.

MLP Perceptron multicouche. Réseau de neurones utilisé pour classifier les lettres statiques à partir des coordonnées de la main.

Modèle 1 Module de vision. Il combine notre modèle custom de détection du haut du corps avec MediaPipe Hands pour produire les points clés du corps et des mains.

Modèle 2 Module de traduction. Il interprète les coordonnées extraites, avec un MLP pour les lettres statiques et un GRU pour les lettres dynamiques.

MSE *Mean Squared Error*. Loss mesurant l'écart quadratique moyen entre une prédiction et une cible.

ODD Objectif de Développement Durable défini par l'ONU.

OKS *Object Keypoint Similarity*. Mesure de similarité entre points clés tenant compte de l'échelle de la personne et de la tolérance des articulations.

OPEX Dépenses d'exploitation, par exemple l'hébergement, l'inférence, la maintenance et la surveillance du service.

Oracle Fournisseur cloud exploré pendant le projet comme piste de déploiement serveur.

Padding Ajout de valeurs ou de marges pour obtenir une taille fixe, par exemple une séquence de longueur constante.

PyTorch Bibliothèque Python utilisée pour développer et entraîner des réseaux de neurones.

Pseudo-labélisation Utilisation d'un modèle existant pour générer automatiquement des annotations qui servent ensuite à entraîner un autre modèle.

ReLU Fonction d'activation qui conserve les valeurs positives et remplace les valeurs négatives par zéro.

ResNet-50 Architecture de réseau de neurones convolutionnel utilisée comme backbone dans la version intégrée de la partie custom du Modèle 1.

Skeleton Représentation simplifiée du corps ou de la main sous forme de points et de segments.

Socket.IO Bibliothèque utilisée pour transmettre les images et résultats en temps réel entre navigateur et serveur.

Soft-Argmax Variante différentiable de l'argmax qui transforme une heatmap en coordonnées par espérance pondérée.

Spatial softmax Méthode transformant une heatmap en distribution de probabilité pour obtenir des coordonnées différentiables.

Transfer learning Réutilisation d'un modèle préentraîné pour réduire les données et le coût nécessaires à un nouvel entraînement.

WebSocket Canal de communication persistant permettant un échange temps réel entre le navigateur et le serveur.

A

Annexe : exemple de structure JSON d'annotation

Cette annexe illustre la structure des fichiers d'annotation utilisés par la chaîne de segmentation. L'exemple provient du fichier `Mariusu_prise1A.json`. Le nom du fichier encode déjà une partie de l'information utile : le signeur, la prise et le label du signe. Ici, `Mariusu_prise1A` désigne une prise de Marius pour la lettre A.

Le fichier présenté est centré sur le suivi des points de la main. Le principe reste le même pour les annotations plus complètes : nous conservons une structure proche de COCO avec des métadonnées, une liste d'images ou de frames, une liste d'annotations et une liste de catégories. Cette organisation facilite le contrôle des données, le chargement automatique par les scripts d'entraînement et l'extension vers des annotations incluant les deux mains et le haut du corps.

```
1 {
2   "info": {
3     "description": "MediaPipe Hands keypoints exported in COCO-style
4     JSON",
5     "source_video": "Mariusu_prise1A.mp4",
6     "fps": 30.13,
7     "frame_count": 88,
8     "width": 1280,
9     "height": 720,
10    "mediapipe_running_mode": "VIDEO "
11  },
12  "images": [
13    {
14      "id": 1,
15      "file_name": "Mariusu_prise1A.mp4#frame_000000",
16      "width": 1280,
17      "height": 720,
18      "frame_index": 0,
19      "time": 0.0
20    }
21  ],
22  "annotations": [
```

```

22  {
23      "id": 1,
24      "image_id": 1,
25      "category_id": 1,
26      "bbox": [488.153, 610.1, 70.269, 109.9],
27      "area": 7722.563,
28      "num_keypoints": 21,
29      "keypoints": [
30          497.478, 610.1, 2,
31          525.218, 616.158, 2,
32          "... autres points de la main ..."
33      ],
34      "handedness": "Right",
35      "score": 0.968634,
36      "track_id": 1
37  },
38  ],
39  "categories": [
40      {
41          "id": 1,
42          "name": "hand",
43          "keypoints": ["wrist", "thumb_cmc", "..."],
44          "skeleton": [[1, 2], [2, 3]]
45      }
46  ]
47  }

```

Listing A.1 – Extrait simplifié d'un fichier COCO adapté pour le suivi de la main.

B

Annexe : questionnaire de validation du besoin

Cette annexe reprend les questions diffusées dans le questionnaire HearMyHands. Ce formulaire avait pour objectif de mieux comprendre les usages de la LSF, les difficultés de traduction et les attentes possibles vis-à-vis d'un outil automatique.

1. Connaissez-vous des moyens de traduction entre la LSF et le français parlé déjà existants ? Si oui, sont-ils efficaces et largement utilisés ?
2. Quelles sont les erreurs souvent commises lors de la traduction de la LSF ? Comment pourrions-nous éviter ces erreurs ?
3. À votre avis, quel serait le support le plus adapté pour un moyen de traduction : une application mobile, un site internet ou un autre support ?
4. Pour commencer notre projet, nous nous focalisons sur les mouvements des mains. Pensez-vous qu'il est intéressant, voire indispensable, de prendre en compte également les mouvements du haut du corps ou du corps entier ?
5. Pensez-vous que la notion de mouvement, et donc le fait de ne pas seulement prendre en compte des images fixes, est indispensable à la traduction de la LSF vers la langue littéraire ?
6. Existe-t-il une sorte d'argot en LSF, avec des mots ou expressions très utilisés par les personnes sourdes, malentendantes ou muettes, mais qui ne se retrouvent pas dans les traductions des discours formels ?
7. Lorsqu'on traduit la LSF en langue parlée, y a-t-il une part d'interprétation liée aux nuances, aux sous-entendus ou aux doubles sens, comme dans la traduction entre langues étrangères ?
8. Existe-t-il des variantes de la LSF selon la région ou les générations, par exemple ?
9. Quel rôle jouent les expressions du visage et les émotions dans la LSF ? Faut-il, selon vous, les prendre en compte dans la traduction ?
10. Un même signe peut-il avoir plusieurs significations différentes selon le contexte ?
11. Quelle est la vitesse de la langue des signes lors d'une discussion normale, par exemple en nombre de signes par seconde ou par minute ?
12. Y a-t-il une manière évidente de repérer la fin d'un signe et le début d'un nouveau ?

C

Annexe : dépôts du projet

Les codes et outils associés au projet sont répartis dans trois dépôts complémentaires. L'application déployée et la page de monitoring sont également accessibles en ligne :

- Application web publique : <https://hearmyhands.asia/>.
- Page de monitoring serveur : <https://hearmyhands.asia/monitor>.
- Dépôt principal : <https://github.com/nmqx/hearmyhands>. Il contient l'application HearMyHands, l'interface, les modules de détection, le MLP et le GRU Ocarina.
- `NarenkuII/Segmentation_Annotation_Tool_for_HMH` : outil de segmentation et d'annotation utilisé pour préparer des clips et des fichiers JSON exploitables.
- `NarenkuII/Dataset_blender_HMH` : dépôt lié aux essais de génération synthétique avec Blender.