VERS UNE ROBOTISATION INDUSTRIELLE AUGMENTÉE PAR LES MODÈLES VISION–LANGAGE–ACTION

Rayen Ghali

THÈSE PRÉSENTÉE À LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET DE LA RECHERCHE EN VUE DE L'OBTENTION DE LA MAITRISE ÈS SCIENCES APPLIQUÉES

FACULTÉ D'INGÉNIERIE
UNIVERSITÉ DE MONCTON
Mai, 2026

Composition du jury

Président du jury :	Gabriel Cormier Ph.D., Professeur titulaire, Faculté d'ingénierie, Université de Moncton
Examinateur externe :	Adel M. Alimi Ph.D., Professeur titulaire, Université de Sfax, ENIS, Tunisie
Examinateur interne :	Mohamed Lamine Faycal Bellaredj Ph.D., Professeur adjoint, Faculté d'ingénierie, Université de Moncton
Directeur de thèse :	Sid Ahmed Selouani Ph.D., Professeur titulaire, Campus de Shippagan, Université de Moncton

Dédicace

Je dédie ce travail à ma mère, qui a tout sacrifié pour me mener là où je suis aujourd'hui. Merci de n'avoir jamais douté de moi, pour ta présence constante et ton soutien indéfectible.

À moi-même, me rappelant de toujours persévérer et de continuer d'avancer.

À ma famille, pour tous ses encouragements ; à tous mes amis, pour leur soutien et leurs mots d'encouragement ; et à toutes les personnes qui m'ont marqué et inspiré tout au long de mon parcours.

Remerciements

En guise de reconnaissance, je tiens à remercier très sincèrement mon directeur de thèse, M. Sid Ahmed Selouani, pour cette opportunité, pour ses précieux conseils et son soutien inestimable, ainsi que pour sa générosité et sa bienveillance, dans le cadre de cette thèse et bien au-delà. Malgré vos nombreuses obligations, vous m'avez honoré par votre suivi attentif tout au long de cette thèse.

Je suis également reconnaissant envers les membres du laboratoire CFRIA-LARIHS pour leur accueil chaleureux et leur soutien tout au long de cette thèse.

Enfin, je tiens à remercier les membres du jury, M. Gabriel Cormier, M. Mohamed Lamine Faycal Bellaredj et M. Adel M. Alimi, pour avoir accepté d'évaluer ce travail.

Sommaire

Ce travail porte sur la conception et la mise en place d'un système autonome de bout en bout pour l'interaction humain-robot, intégrant une interface multimodale permettant une communication vocale bidirectionnelle. Le système proposé permet le contrôle d'un bras robotique prototype SO-101 via un modèle Vision-Language-Action (VLA) généraliste, capable de générer des commandes motrices directement à partir d'observations visuelles et d'instructions en langage naturel.

L'architecture repose sur un agent orchestrateur basé sur un grand modèle de langage (LLM), coordonnant plusieurs modules : reconnaissance vocale, détection d'activité vocale, synthèse vocale et mémoire conversationnelle. Le VLA assure le contrôle robotique en transformant les images des caméras embarquées en actions articulaires. Un module complémentaire de détection de défauts de surface, basé sur une approche SSL-YOLO exploitant l'apprentissage contrastif auto-supervisé, permet l'inspection industrielle en temps réel.

Cette thèse constitue une preuve de concept évaluant les capacités et limitations de l'intégration de modèles d'intelligence artificielle générative dans des tâches robotiques industrielles. Les résultats expérimentaux (notamment un taux de succès de 97% pour le modèle GR00T et une amélioration de 13,2 points en détection few-shot) démontrent la faisabilité de cette approche et ouvrent la voie à de futurs déploiements industriels.

Table des matières

Liste des tableaux

Table des figures

Liste des Abréviations

AAT	Anchor-Aided Training
ACT	Action Chunking with Transformers
AI	Articial Intelligence
API	Application Programming Interface
AP	Average Precision
ASL	Adaptive Support Learning
ASR	Automatic Speech Recognition
BERT	Bidirectional Encoder Representations from Transformers
BPE	Byte-Pair Encoding
CIP	Calcul informatique de pointe
CLAHE	Contrast Limited Adaptive Histogram Equalization
CLIP	Contrastive Language-Image Pre-training
CNN	Convolutional Neural Network
CoT	Chain-of-Thought
DeFS	Decoupled Few-Shot
DETR	DEtection TRansformer
DoF	Degrees of Freedom
EDO	Équation Différentielle Ordinaire
EDS	Équation Différentielle Stochastique
EEF	End-Effector
EEPROM	Electrically Erasable Programmable Read-Only Memory
FFN	Feed-Forward Network
FLARE	Future LAtent Representation Alignment
FLOPs	Floating Point Operations
FOV	Field Of View
FPS	Frames Per Second
FSOD	Few-Shot Object Detection
GELAN	Generalized Efficient Layer Aggregation Network

GPU	Graphics Processing Unit
GPT	Generative Pre-trained Transformer
HBM	High Bandwidth Memory
IoU	Intersection over Union
JIT	Just-In-Time
JSON	JavaScript Object Notation
LLM	Large Language Model
LSTM	Long Short-Term Memory
mAP	mean Average Precision
MLP	Multilayer Perceptron
MRA	Multi-Relational Aggregation
MSE	Mean Squared Error
NMS	Non-maximum suppression
NT-Xent	Normalized Temperature-scaled Cross Entropy
OCR	Optical Character Recognition
PAN	Path Aggregation Network
PEFT	Parameter-Efficient Fine-Tuning
PGI	Programmable Gradient Information
RAG	Retrieval-Augmented Generation
RL	Reinforcement Learning
RLDS	Reinforcement Learning Datasets
RLHF	Reinforcement Learning from Human Feedback
RNN	Recurrent Neural Network
RoPE	Rotary Position Embedding
SigLIP	Sigmoid Loss for Language-Image Pre-training
SSD	Single Shot MultiBox Detector
SSL	Self-Supervised Learning
TTL	Transistor-Transistor Logic
TTS	Text-to-Speech
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
VAD	Voice Activity Detection
ViT	Vision Transformer
VLA	Vision Language Action
VLM	Vision Language Model
VQA	Visual Question Answering
YOLO	You Only Look Once

Introduction générale

Dans l'industrie manufacturière, l'interaction entre opérateurs et robots repose encore largement sur des méthodes traditionnelles : le guidage manuel du bras, des panneaux de contrôle dédiés, ou des séquences d'actions préprogrammées. Ces approches, bien qu'éprouvées en termes de précision et de répétabilité, imposent une rigidité opérationnelle : chaque nouvelle tâche requiert une reprogrammation experte, et toute déviation par rapport au scénario prévu peut compromettre l'exécution. L'émergence récente des Grands Modèles de Langage (LLM) et des modèles Vision-Langage-Action (VLA) ouvre de nouvelles perspectives. Les premiers excellent dans la compréhension du langage naturel et le raisonnement contextuel, tandis que les seconds permettent de générer des commandes motrices directement à partir d'observations visuelles et d'instructions textuelles.

Ce travail présente la conception et l'implémentation d'un système d'interaction humain-robot piloté par la voix, où un opérateur peut commander un bras robotique manipulateur en langage naturel pour exécuter des tâches de manipulation d'objets et d'inspection visuelle. Le système repose sur une architecture à deux niveaux : un agent conversationnel basé sur un LLM assure l'interprétation des requêtes, la planification des actions et l'orchestration des différents modules, tandis qu'un modèle VLA génère les trajectoires motrices nécessaires à l'exécution physique. Cette séparation permet de combiner la flexibilité du raisonnement sémantique avec la réactivité du contrôle neuronal, tout en intégrant des capacités d'inspection spécialisées comme la détection de défauts de surface.

Problématique de recherche

L'intégration des modèles d'intelligence artificielle générative dans les systèmes robotiques industriels soulève des défis fondamentaux à l'intersection de plusieurs domaines. Sur le plan architectural, comment concevoir un système d'orchestration capable d'interpréter des requêtes en langage naturel, de maintenir une cohérence contextuelle et de coordonner l'exécution de tâches hétérogènes ? Sur le plan

du contrôle moteur, les VLA promettent une alternative aux méthodes de programmation traditionnelles, mais quel niveau de performance peuvent-ils atteindre, et quels facteurs (choix architectural, hyperparamètres d'entraînement, qualité et volume des données collectées) déterminent leur précision et leur capacité de généralisation ? Parallèlement, comment doter le système de capacités d'inspection robustes dans des contextes industriels où les données annotées sont rares ? Enfin, est-il possible d'assembler ces composants au sein d'une architecture unifiée fonctionnant avec des latences acceptables pour une interaction naturelle ?

Objectifs de recherche

L'objectif principal de ce travail est de concevoir, implémenter et valider une architecture de système autonome "de bout en bout" qui fusionne l'interaction vocale, le raisonnement par agent et le contrôle robotique neuronal.

Les objectifs spécifiques se déclinent comme suit :

— **Concevoir une architecture d'orchestration** modulaire basée sur des agents LLM, capable de gérer le dialogue, de maintenir un contexte conversationnel et de déléguer dynamiquement des tâches à des outils spécialisés.

— **Intégrer et évaluer des VLA** pour le contrôle moteur, en étudiant la chaîne complète allant de la collecte de données à l'inférence, afin de permettre une manipulation généraliste sans programmation explicite.

— **Développer un module d'inspection industrielle** robuste, capable de détecter des défauts de surface avec une haute précision, en mettant l'accent sur des approches adaptées aux contextes où les données annotées sont rares (few-shot learning).

— **Démontrer la faisabilité d'une interaction multimodale fluide**, où la voix de l'utilisateur pilote directement les actions physiques et les analyses visuelles du robot, validant ainsi le potentiel de ces technologies pour les futurs assistants industriels.

Méthodologie du travail

Afin de répondre à la problématique posée et d'atteindre les objectifs de recherche, la méthodologie adoptée se décline en plusieurs phases clés, allant de l'analyse théorique à la validation expérimentale d'un prototype fonctionnel. Cette démarche

progressive permet de justifier les choix techniques et d'assurer la cohérence de l'ensemble du système.

3.1. Étude et analyse de l'existant

Cette première phase vise à établir un état de l'art solide pour orienter nos choix de conception. Elle consiste en une analyse approfondie de la littérature scientifique et des solutions techniques actuelles, en se concentrant sur :

- les capacités de raisonnement des LLM et les architectures d'agents autonomes ;
- les avancées récentes des VLA et leurs applications en robotique de manipulation ;
- les méthodes de vision par ordinateur pour la détection de défauts, notamment dans des contextes industriels à faibles données (*few-shot*) ;
- les architectures logicielles permettant l'intégration de ces composants hétérogènes en temps réel.

3.2. Conception de l'architecture système

Cette étape définit une structure modulaire et évolutive capable d'orchestrer les différentes intelligences du système. Le travail de conception porte sur :

- la définition d'une stratégie d'orchestration centrale basée sur un LLM pour la compréhension des intentions et la planification ;
- la spécification des interfaces de communication entre les modules (perception, décision, action) ;
- l'élaboration de mécanismes de gestion de la mémoire et du contexte conversationnel pour assurer une interaction fluide.

3.3. Développement du système de contrôle robotique

Cette phase se concentre sur la mise en œuvre de la brique "Action" du système. Elle implique :

- la sélection et la préparation d'une plateforme robotique expérimentale (bras manipulateur) adaptée à la collecte de données ;
- la constitution de jeux de données de démonstrations de qualité pour l'apprentissage par imitation ;

— l'entraînement et l'évaluation comparative de plusieurs architectures de modèles VLA afin d'identifier celle offrant le meilleur compromis entre performance et généralisation ;
— l'exploration de techniques d'optimisation pour améliorer la fluidité et la précision des mouvements générés.

3.4. Développement du module de vision industrielle

Cette étape vise à doter le système de capacités d'inspection visuelle avancées. La méthodologie comprend :

— l'analyse comparative de modèles de détection d'objets de l'état de l'art pour sélectionner une solution adaptée aux contraintes de temps réel ;
— l'étude et l'implémentation d'approches d'apprentissage (telles que l'auto-supervision) permettant de traiter efficacement les classes de défauts rares ;
— l'intégration de ce module comme un outil d'inspection activable à la demande par le système central.

3.5. Intégration et validation expérimentale

La phase finale consiste à assembler les composants développés pour former le système complet. Elle couvre :

— le développement des interfaces d'interaction vocale (reconnaissance et synthèse) ;
— l'intégration technique des modules de contrôle et de vision au sein de l'architecture globale ;
— la validation du système de bout en bout à travers des scénarios d'utilisation concrets, permettant d'évaluer la latence, la robustesse et la pertinence des interactions.

Le système résultant combine ainsi une interface multimodale, un agent orchestrateur intelligent, un contrôleur robotique neuronal et un module d'inspection visuelle, fonctionnant en synergie pour accomplir des tâches complexes.

Contributions de la thèse

La contribution principale de ce travail réside dans la conception et la mise en œuvre d'une architecture modulaire complète pour l'interaction humain-robot autonome, intégrant communication multimodale bidirectionnelle, orchestration

par agent LLM et contrôle robotique par modèle VLA au sein d'un système unifié fonctionnant en boucle fermée. Cette architecture démontre la faisabilité de combiner la planification sémantique de haut niveau, assurée par un LLM capable de raisonner et de prendre des décisions pour automatiser des tâches sans intervention humaine, même dans des environnements changeants, de manière réactive, avec le contrôle moteur de bas niveau généré par un VLA à partir d'observations visuelles et d'instructions en langage naturel.

Une deuxième contribution porte sur l'étude empirique de plusieurs architectures VLA open-source récentes déployées sur une plateforme robotique physique SO-101, analysant leur adaptabilité à différentes tâches de manipulation, environnements et morphologies robotiques (*embodiments*) physiques. Cette étude permet d'identifier leurs performances, leurs limitations et l'impact des hyperparamètres dans des conditions concrètes d'exploitation.

Une troisième contribution concerne l'amélioration des performances des modèles VLA à travers deux axes complémentaires : d'une part, l'optimisation de la qualité des données de démonstration par des techniques de lissage et de filtrage des trajectoires, réduisant le bruit inhérent à la téléopération humaine ; d'autre part, le transfert de connaissances depuis des modèles plus performants vers des architectures plus légères, permettant d'obtenir des politiques de contrôle plus fluides et plus précises tout en maintenant une faible empreinte computationnelle.

Enfin, ce travail intègre un module de détection de défauts de surface basé sur une approche SSL-YOLO exploitant l'apprentissage contrastif auto-supervisé pour améliorer les performances en contexte few-shot, démontrant l'applicabilité du système à des tâches d'inspection industrielle.

Organisation de la thèse

Ce manuscrit est structuré en cinq chapitres. Le chapitre 1 présente un état de l'art des LLM, des modèles Vision-Langage (VLM) et des architectures VLA, permettant de situer notre contribution par rapport aux travaux existants. Le chapitre 2 détaille la conception et l'architecture globale du système d'interaction humain-robot, incluant la description de la plateforme robotique SO-101 et les choix techniques effectués. Le chapitre 3 couvre l'intégration du modèle VLA pour le contrôle robotique, depuis la calibration et la collecte de données jusqu'à l'évaluation comparative des architectures et l'étude des hyperparamètres. Le chapitre 4 présente le module de détection de défauts de surface pour l'inspection industrielle, avec une

comparaison de modèles de détection en temps réel et une approche SSL-YOLO pour le contexte few-shot. Le chapitre 5 détaille l'implémentation des modules de communication multimodale et illustre le fonctionnement de bout en bout du système à travers des scénarios expérimentaux.

Chapitre 1

Cadre théorique des modèles génératifs pour la robotique industrielle

1.1 Introduction

Dans ce premier chapitre, nous présentons le cadre théorique de chaque module utilisé dans notre système. Nous commençons par les modèles de langage de grande taille qui, grâce au mécanisme d'attention, ont révolutionné les interactions humain-intelligence artificielle. Nous poursuivons par les modèles vision-langage, une extension des LLMs qui intègrent la modalité de vision, permettant ainsi à ces modèles d'observer leur environnement et d'acquérir une capacité de raisonnement spatial offrant une compréhension plus riche du monde réel. Nous détaillons ensuite le cadre théorique de la partie centrale de notre recherche qui repose sur ces deux architectures : les modèles vision-langage-action. Cet ajout d'action introduit un nouvel axe de généralisation de ces modèles d'intelligence artificielle, leur permettant d'exécuter des actions et ainsi d'interagir physiquement avec leur environnement. Nous décrivons ensuite les méthodes de vision industrielle traditionnelle, en particulier pour l'inspection de défauts de surface, et nous abordons les défis associés ainsi que leur intégration avec notre système. Nous introduisons par la suite le paradigme d'agent, un paradigme de conception permettant l'autonomie de ces systèmes en leur offrant la capacité d'utiliser des outils externes pour collecter plus d'informations ou exécuter des tâches avant de retourner leur réponse finale. Finalement, nous examinons les approches traditionnelles pour les interfaces humain-robot multimodales dans des contextes industriels.

1.2 Modèles de langage de grande taille (Large Language Models, LLMs)

Un modèle de langage est un système d'intelligence artificielle entraîné à comprendre et à générer du texte. Il apprend, à partir de vastes corpus, à estimer la probabilité du prochain élément d'une séquence (mot ou token), ce qui lui permet de produire des phrases cohérentes. En étant exposé à de grandes quantités de

textes comme des livres, des articles et des pages web, il acquiert progressivement des régularités du langage, notamment la grammaire, le vocabulaire et les relations sémantiques entre les concepts.

1.2.1 Architecture Transformer et mécanisme d'attention

Avant l'émergence des Transformers (Vaswani et coll. 2023) en 2017, les architectures récurrentes telles que les réseaux de neurones récurrents (RNN) (Rumelhart et coll. 1986) et les réseaux à mémoire à long court terme (LSTM) (Graves 2012) dominaient le domaine du traitement du langage naturel. Cependant, ces architectures présentaient des limitations significatives : elles traitaient les séquences de manière séquentielle où chaque étape de temps dépend de la précédente, empêchant la parallélisation efficace et limitant ainsi l'entraînement sur de grandes quantités de données. De plus, lors de la rétropropagation, les gradients se multiplient à travers de nombreuses étapes temporelles, tendant à disparaître ou à exploser exponentiellement, rendant l'apprentissage de patterns éloignés pratiquement impossible.

L'architecture Transformer, introduite par (Vaswani et coll. 2023), a révolutionné le domaine en reposant entièrement sur un mécanisme d'attention plutôt que sur des connexions récurrentes. Au cœur de cette architecture se trouve le mécanisme d'auto-attention (self-attention), qui permet à chaque token d'interagir directement avec tous les autres tokens de la séquence, sans passer par une chaîne de traitements séquentiels. Cette architecture a permis l'émergence de deux paradigmes majeurs : les modèles encodeur comme BERT (Devlin et coll. 2019), qui excellent dans la compréhension du texte (classification, extraction d'information), et les modèles décodeur comme GPT (Radford et coll. 2020), optimisés pour la génération de texte.

a) Le paradigme Requête-Clé-Valeur

Le mécanisme d'attention fonctionne selon un processus de recherche et d'appariement basé sur le paradigme Requête-Clé-Valeur (Q, K, V). Pour une séquence d'entrée donnée, trois projections linéaires distinctes transforment les embeddings d'entrée en vecteurs de requête, de clé et de valeur. Si $\mathbf{x}_i$ représente le i-ème token d'une séquence, alors :

$$\mathbf{q}_i = W^Q \mathbf{x}_i, \quad \mathbf{k}_i = W^K \mathbf{x}_i, \quad \mathbf{v}_i = W^V \mathbf{x}_i \tag{1.1}$$

où W^Q, W^K, et W^V sont des matrices de poids à apprendre.

La requête représente ce qu'un token cible dans la séquence. La clé encode l'information d'un token de manière à la rendre « interrogeable », tandis que la valeur contient le contenu réel exploité par le modèle. Pour une séquence de longueur n, les vecteurs individuels sont empilés en matrices $Q, K, V \in \mathbb{R}^{n \times d_k}$. Les poids d'attention sont calculés via le produit scalaire entre les Queries et les Keys, suivi d'une normalisation softmax :

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{1.2}$$

où $K^\top$ désigne la transposée de la matrice K, et d_k est la dimension des vecteurs de clé (le facteur de normalisation $\sqrt{d_k}$ maintient une variance numérique stable). Cette formulation présente plusieurs avantages majeurs : le calcul des poids d'attention ne dépend pas séquentiellement de la position dans la séquence, permettant une parallélisation complète ; le chemin direct entre chaque paire de tokens élimine le problème d'évanouissement des gradients ; et l'attention capture explicitement les dépendances à long terme sans être limitée par une mémoire cachée de taille fixe.

b) Attention multi-tête

Le mécanisme d'attention multi-tête (multi-head attention) applique l'auto-attention en parallèle avec h ensembles de projections Q, K, V distincts, chacun opérant dans un sous-espace de la dimension du modèle. Si la dimension du modèle est d_{model}, chaque tête opère sur une dimension réduite $d_v = d_{\text{model}}/h$. Les sorties de chaque tête sont concaténées puis projetées via une matrice de sortie :

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{1.3}$$

où chaque $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$, et $W^O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ est la matrice de projection de sortie ($Output$) qui transforme le vecteur concaténé de dimension d_{model} en une représentation finale de même dimension. Cette architecture permet au modèle de capturer différents aspects sémantiques d'une même position.

c) Composants du bloc Transformer

Un bloc Transformer complet intègre plusieurs composants clés. Après la couche d'attention multi-tête, un réseau de neurones feedforward position-wise traite chaque

position indépendamment :

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{1.4}$$

Des connexions résiduelles (He et coll. 2015) contournent chaque sous-couche, permettant aux gradients de circuler directement à travers les couches profondes : output = SubLayer(x) + x. Une normalisation des couches (layer normalization) (Zhang et coll. 2019) stabilise l'entraînement. Cette architecture, répétée sur N niveaux, permet au modèle de construire progressivement des représentations de plus en plus abstraites et contextualisées.

1.2.2 Mécanismes de génération et stratégies de décodage

Après avoir décrit l'architecture interne des Transformers, nous examinons maintenant comment ces modèles produisent effectivement du texte. Les LLM fonctionnent selon un principe auto-régressif : ils construisent le texte séquentiellement, token par token. Un *token* est l'unité atomique de traitement du texte, pouvant correspondre à un mot, une syllabe ou un caractère.

Avant d'être traité par le modèle, le texte brut passe par une étape de *tokenisation*. Le tokenizer segmente le texte en unités discrètes (tokens) et les convertit en identifiants numériques uniques (IDs) correspondant à un index dans le vocabulaire du modèle. Ces IDs sont ensuite projetés dans un espace vectoriel dense via un tableau d'embeddings permettant au modèle de manipuler des représentations sémantiques continues.

Inversement, lors de la génération, le modèle prédit l'ID du prochain token. Le tokenizer effectue alors l'opération inverse, le *detokenization*, en mappant cet ID vers sa représentation textuelle (mot ou sous-mot) et en reconstruisant la phrase complète en concaténant les morceaux, gérant automatiquement les espaces et la ponctuation. Des algorithmes comme Byte-Pair Encoding (BPE) (Sennrich et coll. 2016) ou WordPiece (Song et coll. 2021) sont couramment utilisés pour optimiser ce vocabulaire en équilibrant la taille du vocabulaire et la longueur des séquences.

Formellement, étant donné une séquence de tokens d'entrée $x_1, ..., x_{t-1}$, le modèle estime la distribution de probabilité conditionnelle du prochain token x_t sur l'ensemble du vocabulaire $\mathcal{V}$. Cette distribution est obtenue en appliquant la fonction *Softmax* aux logits z (sorties non normalisées de la dernière couche du réseau) :

$$P(x_t = i | x_{1:t-1}) = \frac{\exp(z_i)}{\sum_{j \in \mathcal{V}} \exp(z_j)} \tag{1.5}$$

Cette approche permet au modèle de capturer les dépendances contextuelles complexes et de générer des réponses syntaxiquement et sémantiquement cohérentes. La figure 1.1 illustre ce processus de prédiction.

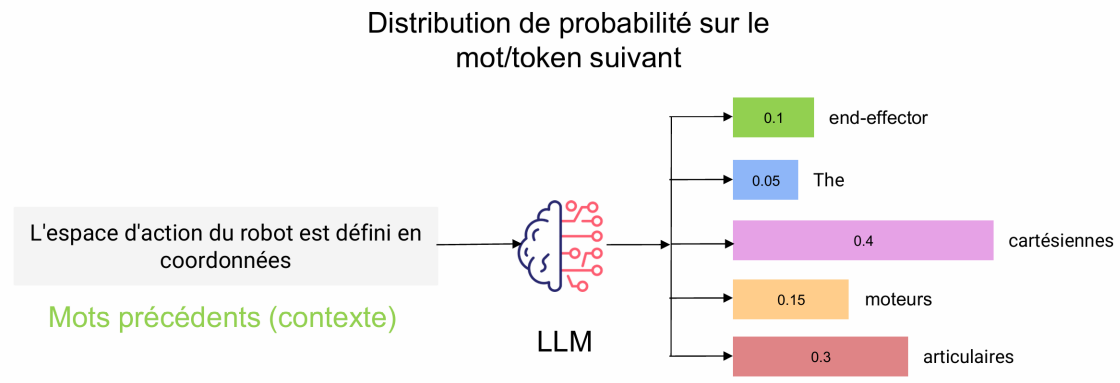


FIGURE 1.1 – Processus de prédiction du prochain token dans un modèle de langage de grande taille (LLM).

Une fois la distribution de probabilité calculée, une stratégie de décodage est nécessaire pour sélectionner le token suivant. C'est ici qu'interviennent des hyperparamètres cruciaux comme la *température*.

La température (T) est un hyperparamètre de mise à l'échelle qui modifie la forme de la distribution de probabilité avant l'échantillonnage. La probabilité ajustée P_i pour le token i devient :

$$P_i = \frac{\exp(z_i/T)}{\sum_{j \in \mathcal{V}} \exp(z_j/T)} \tag{1.6}$$

La figure 1.2 montre l'effet de la température sur la sélection du mot suivant :

L'ajustement de la température permet de moduler le comportement du modèle :

— **Basse température** ($T < 1$) : La distribution devient plus "pointue", favorisant fortement les tokens les plus probables. Cela réduit l'aléatoire et produit des réponses plus déterministes et conservatrices.

— **Haute température** ($T > 1$) : La distribution s'aplatit, augmentant la probabilité relative des tokens moins fréquents. Cela encourage la diversité lexicale et la créativité, au risque d'introduire des incohérences.

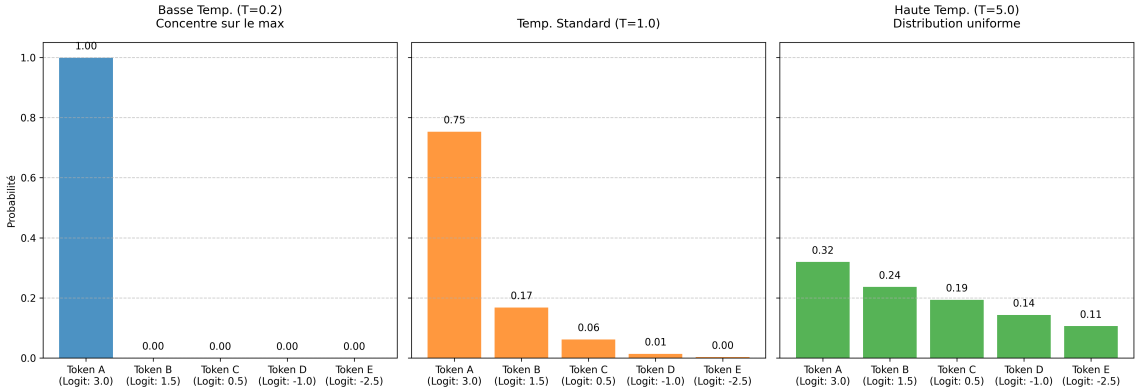


FIGURE 1.2 – Impact de la température sur la distribution de probabilité. Une température basse concentre la probabilité sur les tokens dominants, tandis qu'une température élevée l'uniformise.

1.2.3 Encodage positionnel et RoPE

Pour que les Transformers comprennent l'ordre des tokens, un encodage positionnel est nécessaire. Une innovation majeure, le *Rotary Position Embedding* (RoPE) (SU et coll. 2023), encode l'information positionnelle en appliquant une rotation aux vecteurs Query et Key. Pour un token à la position m et une paire de dimensions $(2i, 2i+1)$, la rotation s'exprime :

$$\begin{pmatrix} q_{2i} \\ q_{2i+1} \end{pmatrix}' = \begin{pmatrix} \cos(m\theta_i) & -\sin(m\theta_i) \\ \sin(m\theta_i) & \cos(m\theta_i) \end{pmatrix} \begin{pmatrix} q_{2i} \\ q_{2i+1} \end{pmatrix} \tag{1.7}$$

où $\theta_i = 10000^{-2i/d_{\text{model}}}$ est la fréquence de base. RoPE offre plusieurs avantages : elle généralise naturellement à des longueurs de séquence non vues lors de l'entraînement, capture implicitement les distances relatives entre tokens, et préserve la norme des vecteurs, stabilisant les calculs numériques.

1.2.4 Lois d'échelle et émergence de capacités

Les *lois d'échelle* (scaling laws) constituent un cadre théorique et empirique fondamental pour comprendre et prédire les performances des modèles de langage. Ces lois établissent que la perte (loss) d'un modèle diminue de manière prévisible selon une loi de puissance en fonction de trois variables principales : le nombre de paramètres N, la quantité de données d'entraînement D (mesurée en tokens), et le budget de calcul C (exprimé en FLOPs) (KAPLAN et coll. 2020 ; HOFFMANN et coll. 2022).

a) Formulation mathématique

La relation entre ces variables et la performance du modèle peut s'exprimer sous plusieurs formes complémentaires. Pour un budget de calcul fixe, la perte optimale suit approximativement :

$$L(C) \propto C^{-\gamma} \tag{1.8}$$

où γ est un exposant empirique. De manière plus détaillée, la perte peut être décomposée en contributions indépendantes :

$$L(N, D) \approx L_0 + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}} \tag{1.9}$$

où L_0 représente la perte irréductible (liée à l'entropie intrinsèque du langage), tandis que les termes A/N^{α} et B/D^{β} capturent respectivement les limitations dues à la taille du modèle et à la quantité de données.

b) Allocation optimale du calcul : la loi de Chinchilla

Les travaux initiaux de Kaplan et coll. (Kaplan et coll. 2020) suggéraient de privilégier l'augmentation des paramètres par rapport aux données d'entraînement. Cependant, l'étude de Hoffmann et coll. (Hoffmann et coll. 2022) a révisé cette perspective en démontrant que les LLM étaient significativement sous-entraînés. En entraînant plus de 400 modèles allant de 70 millions à 16 milliards de paramètres sur des corpus de 5 à 500 milliards de tokens, ils ont établi que, pour un entraînement optimal en termes de calcul, la taille du modèle et le nombre de tokens d'entraînement doivent croître de manière équilibrée : pour chaque doublement de la taille du modèle, le nombre de tokens d'entraînement doit également doubler.

Cette découverte a conduit au modèle Chinchilla (70B paramètres), qui, avec le même budget de calcul que Gopher (280B) mais entraîné sur quatre fois plus de données, a surpassé uniformément des modèles significativement plus grands comme GPT-3 (175B) (Brown et coll. 2020)

c) Émergence de capacités

Un phénomène remarquable lié aux lois d'échelle est l'*émergence* de capacités qualitativement nouvelles au-delà de certains seuils de taille. Contrairement aux améliorations graduelles prédites par les lois de puissance, certaines aptitudes—comme le raisonnement arithmétique complexe, la compréhension d'analogies abstraites, ou le suivi d'instructions multi-étapes—apparaissent

de manière discontinue lorsque le modèle atteint une échelle critique. Ces comportements émergents suggèrent que l'accumulation quantitative de paramètres et de données peut induire des transitions qualitatives dans les capacités du modèle.

d) Au-delà de l'échelle brute

Si les lois d'échelle ont guidé la course vers des modèles toujours plus grands, des recherches récentes démontrent que l'échelle n'est pas le seul levier d'amélioration. Des techniques d'optimisation architecturale, de distillation de connaissances (Hinton et coll. 2015), et d'entraînement sur des données de haute qualité permettent à des modèles plus compacts d'atteindre des performances comparables à des modèles plusieurs fois plus grands. Cette tendance vers l'efficacité a favorisé l'émergence de modèles open-source performants, rendant les capacités avancées des LLM accessibles à une communauté élargie.

1.2.5 Modèles open-source : étude de cas Gemma

La démocratisation des LLM a été considérablement accélérée par la publication de modèles open-source performants tels que LLaMA (Touvron et coll. 2023) (Meta), Qwen (Bai et coll. 2023a) (Alibaba) et Gemma (Team et coll. 2024) (Google). Nous détaillons ici **Gemma 2B**, un modèle représentatif qui servira de composant linguistique dans les architectures multimodales présentées ultérieurement (Section 1.3.2). Ce décodeur Transformer comprend 2,51 milliards de paramètres (dont 1,98 milliard non-embedding), 18 couches avec une dimension $d_{\text{model}} = 2048$, une dimensionnalité des clés $d_k = 256$, et 8 têtes d'attention.

a) Attention multi-requête

Gemma 2B utilise l'*attention multi-requête* (Multi-Query Attention) (Shazeer 2019), où une seule tête gère les projections de clé et valeur partagées, tandis que les requêtes restent répliquées sur plusieurs têtes :

$$\text{MultiQuery}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{1.10}$$

Cette configuration réduit considérablement le nombre de paramètres et les besoins en mémoire, permettant un déploiement sur des appareils aux ressources limitées.

b) Innovations architecturales

Gemma 2B incorpore plusieurs innovations techniques. La normalisation **RMSNorm** (Zhang et coll. 2019) simplifie la layer normalization en se concentrant uniquement sur l'amplitude :

$$\text{RMSNorm}(x) = \gamma \frac{x}{\sqrt{\frac{1}{d}\sum_i x_i^2 + \epsilon}} \tag{1.11}$$

où γ est un paramètre appris de mise à l'échelle, d est la dimension de l'entrée x, et ϵ est une faible constante (typiquement 10^{-6}) ajoutée au dénominateur pour assurer la stabilité numérique et éviter les divisions par zéro.

Les activations **GeGLU** (Shazeer 2020) remplacent ReLU par un mécanisme de gating :

$$\text{GeGLU}(x) = \text{GELU}(xW_g + b_g) \odot (xW + b) \tag{1.12}$$

où $W_g \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ et $b_g \in \mathbb{R}^{d_{\text{out}}}$ sont respectivement la matrice de poids et le biais du réseau de gating, $W \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ et $b \in \mathbb{R}^{d_{\text{out}}}$ sont les poids et biais de la transformation principale, GELU est la fonction d'activation *Gaussian Error Linear Unit* qui applique une approximation régulière de la fonction d'échelon pour laisser passer progressivement les valeurs positives, et $\odot$ désigne la multiplication élément par élément. Cette architecture permet au réseau de sélectionner dynamiquement quels chemins d'information propager.

Le modèle utilise également une alternance d'attention à fenêtre glissante locale et d'attention globale, ainsi que le *logit soft-capping* pour stabiliser l'entraînement.

1.3 Modèles vision-langage (VLMs)

Les VLM étendent les capacités des LLM (Section 1.2) au domaine multimodal, fusionnant compréhension visuelle et traitement du langage naturel (Radford et coll. 2021 ; Jia et coll. 2021). L'évolution de ces modèles a suivi plusieurs innovations majeures.

La première génération de VLM utilise l'apprentissage contrastif à grande échelle, établi par CLIP (*Contrastive Language-Image Pre-training*) et ALIGN (Radford et coll. 2021 ; Jia et coll. 2021). Ces modèles exploitent des paires image-texte disponibles sur le web pour apprendre un espace de représentation aligné, où les images et leurs descriptions textuelles correspondantes sont proches dans l'espace d'embedding. L'approche contrastive utilise une fonction de perte basée sur le

softmax normalisé, qui nécessite une vue globale de toutes les similarités par paire dans un mini-batch.

La deuxième génération a évolué vers des architectures encoder-decoder génératives, s'inspirant du succès des modèles T5 en traitement du langage naturel (Raffel et coll. 2023). Des travaux tels que PaLI (Chen et coll. 2023b) ont démontré que la combinaison d'encodeurs visuels de grande taille avec des modèles de langage encoder-decoder permettait d'atteindre des performances remarquables sur une diversité de tâches multimodales, y compris la génération de texte et la réponse à des questions visuelles (*Visual Question Answering*) (Agrawal et coll. 2016). Les versions ultérieures, PaLI-X et PaLI-3, ont continué à repousser les limites avec des modèles plus grands et des stratégies d'entraînement affinées.

Parallèlement, l'introduction d'architectures decoder-only autorégressives a transformé le paradigme. Des modèles comme LLaVA (Liu et coll. 2023b) ont montré que l'utilisation de décodeurs autorégressifs seuls, combinés à des encodeurs visuels, permettait d'obtenir des résultats compétitifs tout en réduisant la complexité architecturale.

1.3.1 Architecture des modèles vision-langage

L'architecture standard d'un VLM moderne se compose de trois composants : un encodeur visuel, un décodeur linguistique LLM, et une couche de projection reliant les deux modalités.

a) L'encodeur visuel

L'encodeur visuel transforme une image brute en une séquence de tokens visuels exploitables par le modèle de langage. La majorité des VLMs s'appuient sur l'architecture Vision Transformer (ViT) (Dosovitskiy et coll. 2021), qui applique le mécanisme d'attention au domaine visuel.

Le processus de traitement d'une image par un ViT commence par la division de l'image en patches non-chevauchants de taille fixe. Pour une image de résolution $H \times W$ divisée en patches de taille $P \times P$, le nombre de patches est $\frac{H}{P} \times \frac{W}{P}$. Chaque patch est ensuite aplati en un vecteur et projeté linéairement dans l'espace d'embedding du modèle pour produire un token de dimension d. Un token de classe (CLS) spécial est ajouté au début de la séquence, portant le nombre total de tokens à $\frac{HW}{P^2} + 1$. Des embeddings positionnels sont également ajoutés pour préserver

l'information spatiale, permettant au modèle de comprendre la disposition relative des différentes régions de l'image.

L'encodeur applique ensuite l'auto-attention multi-tête sur l'ensemble de ces tokens. Contrairement aux réseaux de neurones convolutifs, le ViT calcule une matrice d'attention globale où chaque token peut porter attention à tous les autres tokens. Cette approche offre une meilleure modélisation contextuelle mais entraîne une complexité quadratique $\mathcal{O}(n^2)$ en fonction du nombre de tokens.

b) Le connecteur : alignement des modalités

Une fois les caractéristiques visuelles extraites, elles doivent être projetées dans l'espace d'embedding du modèle de langage. Dans les architectures denses comme PaliGemma, les embeddings visuels sont projetés via une couche linéaire dans l'espace dimensionnel du LLM, puis concaténés en préfixe des tokens textuels (Beyer et coll. 2024).

D'autres modèles comme Flamingo (Alayrac et coll. 2022) séparent les modalités et utilisent des couches d'**attention croisée**. Contrairement à l'auto-attention où les requêtes, clés et valeurs proviennent de la même séquence, l'attention croisée utilise les tokens d'une modalité (le texte) comme requêtes (*Query*) pour interroger les tokens d'une autre modalité (la vision) servant de clés et valeurs (*Key/Value*). Ce mécanisme permet au décodeur linguistique d'intégrer dynamiquement les informations visuelles lors de la génération.

c) Mécanismes d'Attention dans les VLMs

Les VLMs emploient différents types de mécanismes d'attention selon leur objectif et leur architecture.

- **Attention auto-régressive (causale)** : Typique des décodeurs de texte, elle contraint chaque position t à ne porter attention qu'aux positions précédentes $t' \leq t$ (Vaswani et coll. 2023). Mathématiquement, cela se traduit par une matrice de masque triangulaire inférieure ($M_{ij} = 0$ si $i < j$). Cette configuration force une génération séquentielle où chaque token est prédit conditionnellement à l'historique.
- **Attention bidirectionnelle** : Employée dans les encodeurs visuels (comme CLIP et SigLIP (*Sigmoid Loss for Language-Image Pre-training*)) et les architectures encoder-decoder, elle permet à chaque token de porter attention à tous les autres tokens sans restriction. Cela facilite une capture globale des

relations contextuelles en un seul passage pour extraire des représentations visuelles riches (Radford et coll. 2021 ; Zhai et coll. 2023).

— **Schéma hybride** : Ce schéma applique une attention bidirectionnelle sur les tokens d'image et de l'entrée textuelle (préfixe), et une attention causale sur la réponse générée (suffixe). Cette stratégie permet aux tokens d'image de tirer profit d'une "prévoyance" sur la tâche à accomplir (spécifiée dans le préfixe) avant d'entamer la génération de la réponse.

d) Génération auto-régressive

Les VLMs héritent du mécanisme de génération auto-régressive des LLMs (Section 1.2) : le modèle prédit le token suivant conditionnellement à l'ensemble des tokens précédents (visuels et textuels). La perte de prédiction du prochain token est appliquée uniquement sur les tokens de sortie.

Cette interface « image+texte → texte » permet de traiter les tâches de vision par ordinateur (classification, description, VQA (*Réponse aux Questions Visuelles*)) ainsi que des tâches structurées (détection, segmentation) en convertissant les sorties en format textuel via des tokens spécialisés (Beyer et coll. 2024).

1.3.2 Étude de cas : PaliGemma

Plusieurs VLMs ouverts ont émergé ces dernières années, notamment LLaVA (Liu et coll. 2023b) ou Qwen-VL (Bai et coll. 2023b), chacun proposant des approches distinctes pour l'intégration vision-langage. PaliGemma illustre particulièrement bien comment des choix architecturaux judicieux permettent d'atteindre des performances remarquables avec un budget paramétrique modeste.

PaliGemma combine l'encodeur visuel SigLIP avec le modèle de langage Gemma-2B (Section 1.2.5) pour former un VLM de moins de 3 milliards de paramètres. Malgré cette taille modeste, il atteint des performances comparables à des modèles dix à cent fois plus grands sur plus de 40 benchmarks.

a) SigLIP : un encodeur visuel contrastif optimisé

L'encodeur visuel de PaliGemma repose sur SigLIP, une évolution majeure des modèles contrastifs vision-langage comme CLIP (Zhai et coll. 2023). Cette architecture sera également adoptée par plusieurs modèles VLA présentés en Section 1.4. La différence fondamentale avec CLIP réside dans la fonction de perte utilisée lors du pré-entraînement contrastif.

i. Limitations de la perte softmax de CLIP CLIP utilise une perte softmax contrastive qui compare chaque paire image-texte à toutes les autres paires du batch (RADFORD et coll. 2021). Cette approche nécessite une normalisation globale sur l'ensemble du mini-batch, impliquant une communication coûteuse entre tous les processeurs lors de l'entraînement distribué. De plus, la qualité des représentations apprises dépend fortement de la taille du batch : des batchs plus grands fournissent plus de négatifs, améliorant l'apprentissage contrastif, mais augmentant proportionnellement les besoins en mémoire et en synchronisation.

ii. La perte sigmoïde de SigLIP SigLIP reformule l'apprentissage contrastif comme un ensemble de classifications binaires indépendantes. Au lieu de comparer une paire positive à toutes les autres paires du batch via une normalisation softmax, SigLIP applique une fonction sigmoïde séparément à chaque paire image-texte, déterminant si elle constitue une correspondance (label $y_{ij} = 1$) ou non (label $y_{ij} = -1$) (ZHAI et coll. 2023)

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N}\log\sigma\left(y_{ij}\cdot z_i\cdot t_j\right) \tag{1.13}$$

où z_i et t_j désignent respectivement les embeddings L2-normalisés de la i-ème image et du j-ème texte, $\sigma(x) = 1/(1+e^{-x})$ est la fonction sigmoïde, et $y_{ij} = 1$ si $i = j$ (paire positive), $y_{ij} = -1$ sinon (paire négative). Cette formulation traite chaque paire comme une décision binaire indépendante : pour les paires positives, la perte encourage un produit scalaire élevé ; pour les paires négatives, elle pénalise les produits scalaires positifs.

L'avantage clé de cette approche réside dans l'absence de dépendance globale entre les paires. Contrairement au softmax qui nécessite de calculer un dénominateur sur l'ensemble du batch, la sigmoïde évalue chaque paire isolément. Cette propriété confère plusieurs bénéfices pratiques :

— **Passage à l'échelle efficace** : l'entraînement distribué ne requiert plus de synchronisation globale pour la normalisation. Chaque processeur peut calculer la perte sur ses paires locales indépendamment, permettant des batchs atteignant un million d'exemples.

— **Robustesse aux petits batchs** : contrairement à CLIP où la qualité des représentations dépend fortement du nombre de négatifs dans le batch, SigLIP maintient des performances stables même avec des batchs de taille modeste (quelques milliers d'exemples).

— **Réduction de l'empreinte mémoire** : la complexité mémoire passe de $\mathcal{O}(B^2)$ à $\mathcal{O}(b^2)$, où B représente la taille du batch global et b la taille du batch local par dispositif. Cette réduction découle directement de l'absence de matrice de similarité globale.

iii. Architecture ViT-So400m PaliGemma utilise la variante « shape-optimized » de SigLIP à 400 millions de paramètres (ViT-So400m). Cette architecture résulte d'une optimisation systématique des hyperparamètres du Vision Transformer — profondeur du réseau, dimension des embeddings, nombre de têtes d'attention — guidée par des lois d'échelle (*scaling laws*) pour maximiser les performances sous contrainte d'un budget paramétrique fixe de 400M (Zhai et coll. 2023). Le modèle traite des images de 224×224 pixels divisées en patches de 14×14, produisant une grille de $16 \times 16 = 256$ tokens visuels. Malgré sa taille modeste, ce modèle rivalise avec des encodeurs comme ViT-G (1.8B) grâce à l'efficacité combinée de la perte sigmoïde et de l'optimisation architecturale.

b) Intégration multimodale et schéma d'attention Prefix-LM

L'intégration entre l'encodeur SigLIP et le décodeur Gemma-2B s'effectue via une projection linéaire simple. Les embeddings visuels sont projetés linéairement vers l'espace dimensionnel de Gemma, privilégiant une transmission dense de l'information visuelle.

i. Construction de la séquence d'entrée La séquence de tokens d'entrée est structurée comme suit :

$$\text{tokens} = [\text{img}_1, \ldots, \text{img}_{N_{\text{img}}}, \text{BOS}, \text{pref}_1, \ldots, \text{pref}_{N_{\text{pref}}}, \backslash\text{n}, \text{suff}_1, \ldots, \text{suff}_{N_{\text{suff}}}, \text{EOS}] \tag{1.14}$$

où N_{img} désigne le nombre de tokens visuels (256, 1024 ou 4096 selon la résolution), BOS (*Beginning of Sequence*) marque le début de la séquence textuelle, le caractère de nouvelle ligne (\n) sert de séparateur entre le préfixe (la requête) et le suffixe (la réponse), et EOS (*End of Sequence*) signale la fin de la génération. Le préfixe contient la requête textuelle de l'utilisateur, tandis que le suffixe correspond à la réponse générée par le modèle.

PaliGemma adopte le schéma d'attention hybride Prefix-LM décrit précédemment : l'attention bidirectionnelle s'applique sur les tokens image et préfixe, permettant une contextualisation complète avant la génération, tandis que

l'attention causale gouverne la génération du suffixe. Cette configuration améliore empiriquement les performances par rapport à un masquage purement autorégressif.

ii. Entraînement conjoint de l'encodeur visuel Contrairement à la pratique courante qui consiste à geler l'encodeur visuel pendant l'entraînement multimodal, PaliGemma entraîne conjointement SigLIP et Gemma. Les auteurs démontrent que cette approche améliore significativement les capacités de compréhension spatiale et relationnelle, des aspects où les modèles contrastifs présentent traditionnellement des lacunes.

c) Stratégie d'entraînement multi-étapes

L'entraînement de PaliGemma suit un protocole progressif en quatre étapes :

i. Étape 1 : Pré-entraînement unimodal Les deux composants du modèle sont d'abord pré-entraînés indépendamment sur leurs modalités respectives : l'encodeur visuel SigLIP apprend à partir de 40 milliards de paires image-texte, tandis que le modèle de langage Gemma-2B est pré-entraîné sur du texte généraliste. Cette séparation permet d'exploiter des méthodes d'entraînement éprouvées pour chaque modalité.

ii. Étape 2 : Fusion multimodale Les deux composants sont assemblés et entraînés conjointement sur un ensemble diversifié de tâches *task mixture* vision-langage comprenant environ 1 milliard d'exemples à résolution 224×224 avec une longueur de séquence de 128 tokens. Cet ensemble inclut : la génération de descriptions d'images, la reconnaissance de texte dans les images (*Optical Character Recognition*, OCR), la réponse à des questions visuelles, et la détection d'objets. Chaque type de tâche est préfixée par un identifiant unique (`ocr`, `detect`, etc.) pour éviter les conflits de signal d'apprentissage entre compétences.

iii. Étape 3 : Adaptation aux hautes résolutions Le modèle est progressivement adapté à traiter des images de plus haute résolution : d'abord 448×448 pixels, puis 896×896 pixels. Les tâches nécessitant des détails fins, comme la reconnaissance de texte, bénéficient d'une priorité accrue durant cette phase. Cette étape produit trois versions du modèle de base optimisées pour différentes résolutions (224px, 448px, 896px).

iv. Étape 4 : Spécialisation pour les applications cibles Les modèles de base sont finalement spécialisés pour des tâches spécifiques par ajustement fin (*fine-tuning*), où tous les paramètres du modèle continuent d'être entraînés sur des données propres à l'application visée. Cette approche permet d'atteindre des performances de pointe sur plus de 40 benchmarks (Beyer et coll. 2024).

Les VLM comme PaliGemma démontrent l'efficacité de l'intégration vision-langage pour des tâches de compréhension et de génération. L'étape suivante consiste à étendre ces modèles vers la génération d'actions, permettant ainsi le contrôle moteur direct et l'interaction physique avec l'environnement.

1.4 Modèles vision-langage-action (VLAs)

Les VLA étendent les capacités des VLM en ajoutant la génération d'actions robotiques (Kim et coll. 2024). Contrairement aux VLM qui se limitent à la compréhension visuelle et linguistique, les VLA relient directement la perception multimodale au contrôle moteur, permettant aux robots d'interpréter des instructions en langage naturel et des observations visuelles pour générer des séquences d'actions continues (Zitkovich et coll. 2023). Ces modèles intègrent trois composantes fondamentales : l'encodage visuel pour la compréhension des scènes robotiques, le traitement du langage naturel pour la transmission d'instructions sémantiques, et la génération d'actions bas-niveau exécutables sur les robots.

1.4.1 Méthodes de génération d'actions

Un aspect fondamental des VLA réside dans la manière dont ils produisent les commandes motrices à partir des représentations apprises. Trois approches principales se distinguent, chacune offrant des compromis différents entre rapidité d'exécution, précision des mouvements et complexité de mise en œuvre.

a) Approche autorégressive et ses limites

L'approche la plus intuitive consiste à générer les actions une par une, de manière séquentielle, à l'image de la génération de texte dans les LLM classiques. Cependant, cette méthode présente plusieurs limitations pour le contrôle robotique :

— **Latence élevée** : produire une séquence de T actions nécessite T passes successives dans le réseau, ce qui crée un délai incompatible avec le contrôle en temps réel à haute fréquence.

— **Perte de précision** : les mouvements robotiques sont naturellement continus, mais cette approche impose de les convertir en catégories discrètes (tokens), introduisant une approximation.
— **Accumulation d'erreurs** : lors de la génération, chaque nouvelle action dépend des prédictions précédentes plutôt que des vraies actions, amplifiant progressivement les imprécisions — un phénomène connu sous le nom de biais d'exposition (*exposure bias*).

Face à ces limitations, deux alternatives génèrent l'*intégralité de la séquence d'actions en une seule passe*, offrant à la fois un parallélisme de calcul et une représentation continue des mouvements.

b) Politique de diffusion

La politique de diffusion (Chi et coll. 2024) adapte les modèles génératifs par diffusion — initialement conçus pour la synthèse d'images — à la génération de trajectoires robotiques. Le principe repose sur un processus de débruitage progressif : partant d'un signal complètement aléatoire, le modèle apprend à retrouver pas à pas une trajectoire d'actions valide.

i. Phase d'entraînement Durant l'entraînement, le modèle apprend à inverser un processus de corruption du signal :

1. **Corruption progressive** : une séquence d'actions $\mathbf{a}_0$ issue des démonstrations humaines est progressivement dégradée par ajout de bruit gaussien sur K étapes successives. À chaque niveau $k \in \{1, \ldots, K\}$, on obtient une version de plus en plus bruitée $\mathbf{a}_k$.
2. **Apprentissage du débruitage** : le réseau de neurones ϵ_θ (paramétré par θ) apprend à estimer le bruit ϵ qui a été ajouté, en fonction de trois entrées : la séquence bruitée $\mathbf{a}_k$, le niveau de bruit k, et l'observation courante o (image de la caméra du robot).
3. **Fonction de coût** : l'entraînement utilise l'erreur quadratique moyenne (MSE - Mean Squared Error) pour minimiser l'écart entre le bruit réel ϵ et le bruit prédit $\hat{\epsilon} = \epsilon_\theta(\mathbf{a}_k, k, o)$.

ii. Phase d'inférence Pour générer une nouvelle trajectoire à partir d'une observation o, le processus inverse est appliqué :

1. **Point de départ** : on échantillonne un vecteur de bruit pur $\mathbf{a}_K$ suivant une distribution gaussienne standard $\mathcal{N}(0, \mathbf{I})$, où $\mathbf{I}$ désigne la matrice identité.

2. **Débruitage itératif** : pour chaque niveau k décroissant de K à 1, le réseau estime le bruit présent dans $\mathbf{a}_k$ et le soustrait partiellement, produisant une version moins bruitée $\mathbf{a}_{k-1}$.
3. **Résultat** : après K itérations (typiquement 50 à 100), on obtient la séquence d'actions finale $\mathbf{a}_0$, prête à être exécutée.

Mathématiquement, ce processus correspond à la résolution d'une équation différentielle stochastique (EDS), ce qui confère une nature probabiliste aux trajectoires générées : deux exécutions peuvent produire des trajectoires légèrement différentes (Chi et coll. 2024).

c) *Flow Matching*

Le *Flow Matching* (Lipman et coll. 2023) propose une alternative plus directe et efficace. Au lieu d'apprendre à débruiter progressivement, cette méthode apprend un **champ de vitesse** qui transporte le bruit vers les données cibles en suivant un chemin rectiligne.

i. Phase d'entraînement L'apprentissage s'effectue en trois étapes :

1. **Définition du chemin** : pour chaque paire composée d'un bruit initial $\mathbf{a}_0 \sim \mathcal{N}(0, \mathbf{I})$ et d'une trajectoire cible $\mathbf{a}_1$ issue des démonstrations, on définit un chemin linéaire $\mathbf{a}_t = (1-t)\mathbf{a}_0 + t\mathbf{a}_1$ paramétré par le temps $t \in [0, 1]$.
2. **Apprentissage de la direction** : le réseau v_θ apprend à prédire le vecteur de vitesse $(\mathbf{a}_1 - \mathbf{a}_0)$ permettant de se déplacer du bruit vers la cible, en fonction de la position courante $\mathbf{a}_t$, du temps t, et de l'observation o.
3. **Fonction de coût** : l'entraînement utilise l'erreur quadratique moyenne (MSE - Mean Squared Error) pour minimiser l'écart entre la vitesse prédite $\hat{v} = v_\theta(\mathbf{a}_t, t, o)$ et la vitesse théorique $(\mathbf{a}_1 - \mathbf{a}_0)$.

ii. Phase d'inférence La génération d'une trajectoire suit un processus d'intégration numérique :

1. **Point de départ** : on échantillonne un vecteur de bruit $\mathbf{a}_0 \sim \mathcal{N}(0, \mathbf{I})$.
2. **Transport par le flux** : en utilisant un solveur d'équations différentielles ordinaires (EDO) comme la méthode d'Euler ou de Runge-Kutta, on intègre le champ de vitesse v_θ de $t = 0$ à $t = 1$ en N pas discrets.
3. **Résultat** : après N itérations (typiquement 5 à 10 seulement), on obtient la trajectoire finale $\mathbf{a}_1$.

Cette approche présente deux avantages majeurs. Premièrement, les chemins rectilignes permettent au solveur d'utiliser des pas d'intégration plus grands, réduisant considérablement le nombre d'itérations nécessaires (Lipman et coll. 2023). Deuxièmement, la génération est déterministe : contrairement à la diffusion, une fois le bruit initial fixé, le résultat est entièrement reproductible.

d) Bilan comparatif des trois approches

Le tableau 1.1 récapitule les caractéristiques distinctives de chaque méthode de génération d'actions.

Tableau 1.1 – Comparaison des trois méthodes de génération d'actions pour les VLA : l'approche autorégressive génère les actions séquentiellement, tandis que la diffusion et le *Flow Matching* produisent la séquence complète en parallèle.

Aspect	**Autoregressif**	**Diffusion**	***Flow Matching***
Mécanisme	Prédiction du token suivant	Débruitage itératif	Régression d'un champ de vitesse
Type d'équation	—	EDS (stochastique)	EDO (déterministe)
Sortie du réseau	$\hat{a}_t$ (prochain token)	$\hat{\epsilon}$ (bruit estimé)	$\hat{v}$ (vitesse estimée)
Itérations requises	T (longueur de la séquence)	50–100	5–10
Format de sortie	Valeurs discrètes	Valeurs continues	Valeurs continues

Le *Flow Matching* offre un compromis particulièrement adapté au contrôle robotique en temps réel : il préserve la capacité à générer des trajectoires continues tout en réduisant drastiquement le nombre d'itérations nécessaires. Cette efficacité explique son adoption croissante dans les VLA récents.

1.4.2 Jeux de données de pré-entraînement multi-morphologies *multi-embodiments*

Le succès des VLA, notamment leur capacité à s'adapter rapidement à des domaines hors-distribution, repose sur un pré-entraînement avec des jeux de données robotiques vastes et diversifiés couvrant un large éventail d'architectures robotiques et de scénarios de tâches (Collaboration et coll. 2024). Un défi majeur est l'hétérogénéité que ces modèles doivent gérer, depuis les configurations matérielles jusqu'aux stratégies de collecte de données. Cette hétérogénéité se manifeste non seulement dans les espaces d'actions spécifiques à chaque morphologie

robotique (*embodiment*), mais aussi dans les variations de configuration telles que les paramètres de caméra, les domaines visuels et les distributions de tâches (Zheng et coll. 2025).

Open-X Embodiment (Collaboration et coll. 2024), publié en octobre 2023, représente l'initiative la plus ambitieuse pour centraliser et standardiser les données robotiques multimodales. Ce dépôt collaboratif agrège plus d'un million de trajectoires réelles provenant de 22 robots distincts, collectées dans 21 institutions internationales. Le dataset comprend 60 datasets individuels convertis au format standardisé RLDS (Robot Learning Data Standard). Il couvre 527 compétences réparties sur 160 266 tâches, incluant des primitives de manipulation variées (saisie, déplacement, poussée, placement, navigation, ouverture, fermeture, insertion, assemblage) appliquées à un spectre complet d'objets domestiques.

DROID (Distributed Robot Interaction Dataset) (Khazatsky et coll. 2025), introduit en mars 2024, met l'accent sur la diversité géographique et la collecte systématique à grande échelle. Avec 76 000 trajectoires représentant 350 heures d'interaction robotique, DROID couvre 564 scènes et 86 tâches collectées par 50 opérateurs dispersés en Amérique du Nord, Asie et Europe sur 12 mois. La diversité du dataset réside dans son approche de collecte distribuée : en changeant de scènes environ toutes les 20 minutes et en laissant aux collecteurs la liberté de choisir les tâches, l'ensemble de données capture une variation remarquable en environnements naturels (cuisines, chambres, laboratoires, bureaux). Les résultats empiriques démontrent que l'entraînement avec DROID conduit à des politiques avec une capacité de généralisation améliorée.

Plusieurs autres datasets complètent cet écosystème : **BridgeData v2** (Walke et coll. 2024) fournit 60 096 trajectoires de manipulation collectées dans 24 environnements distincts ; **AgiBot World Colosseo** (AgiBot-World-Contributors et coll. 2025) plus d'un million de trajectoires couvrant 217 tâches avec 731 instances d'objets annotées sémantiquement ; **RoboMIND** (Wu et coll. 2025) privilégie la qualité avec 107 000 trajectoires couvrant 479 tâches sur 4 morphologies robotiques distinctes, chaque démonstration étant enrichie d'observations multi-vues et de descriptions textuelles détaillées.

1.4.3 Protocoles d'évaluation standardisés

L'évaluation rigoureuse des VLA requiert des benchmarks standardisés permettant la comparaison équitable des architectures. **LIBERO** (Liu et coll. 2023a) constitue le benchmark de référence pour l'évaluation en

manipulation longue-horizon, utilisant un bras robotique Franka Panda simulé dans l'environnement MuJoCo. Il propose 130 tâches organisées en cinq suites de complexité progressive (LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, LIBERO-Long et LIBERO-90), évaluant respectivement le raisonnement spatial, la manipulation d'objets variés, l'atteinte d'objectifs multiples, les séquences longues et la généralisation inter-tâches (Liu et coll. 2023a). Les métriques évaluent principalement le taux de succès sur chaque suite (Mao et coll. 2025). **CALVIN** (Mees et coll. 2022) se focalise sur l'apprentissage de tâches conditionnées par langage naturel, utilisant un bras Franka Panda avec pince parallèle dans un environnement de bureau simulé (PyBullet). Le benchmark évalue la capacité à enchaîner jusqu'à 5 sous-tâches consécutives et teste explicitement la généralisation zéro-shot sur des instructions novatrices. **Meta-World** (Yu et coll. 2021) fournit un benchmark standardisé avec 50 tâches de manipulation (ML50) sur un bras Sawyer simulé dans MuJoCo, conçu initialement pour le méta-apprentissage et l'apprentissage par renforcement multi-tâches. Il demeure pertinent pour évaluer l'adaptation rapide via fine-tuning et la généralisation à de nouvelles configurations d'objets. Enfin, **VLABench** (Zhang et coll. 2024) représente l'effort le plus ambitieux avec 100 catégories de tâches et plus de 2000 objets dans un environnement simulé flexible. Il évalue la compréhension 3D, le raisonnement spatial, le suivi d'instructions complexes en langage naturel non-templateisé et le raisonnement compositionnel.

1.4.4 Évolution des VLA

Comme l'illustre la figure 1.3, l'histoire des VLA débute avec des approches pionnières propriétaires développées par Google DeepMind avant de progresser vers un écosystème ouvert grâce à la création de l'ensemble de données Open X-Embodiment.

a) RT-1

RT-1 (Brohan et coll. 2023), présenté en décembre 2022 par Google DeepMind, constitue la première démonstration réussie à grande échelle de l'approche VLA. Entraîné sur 130 000 trajectoires couvrant plus de 700 tâches collectées sur 17 mois avec une flotte de 13 robots Everyday Robots (bras 7-DOF, pince deux doigts, base mobile), RT-1 (35 millions de paramètres) adopte une architecture transformer basée sur la tokenisation, combinant un EfficientNet-B3 pré-entraîné avec un TokenLearner

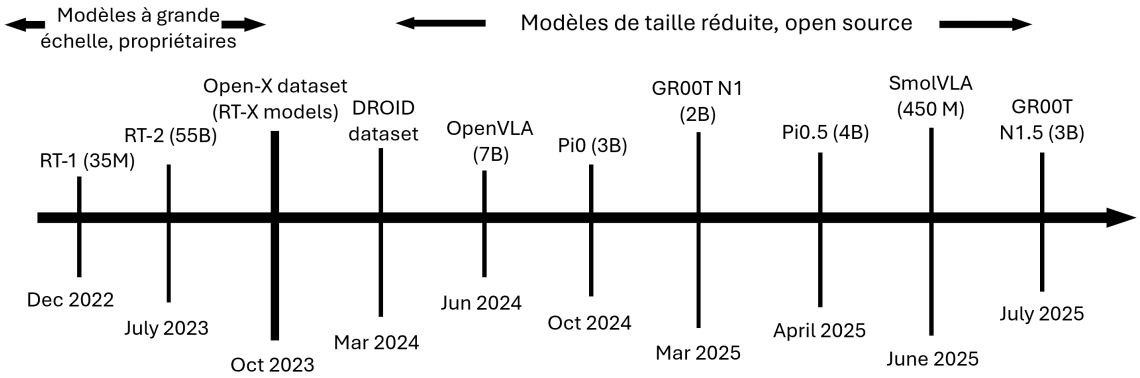


FIGURE 1.3 – Évolution chronologique des modèles VLA.

pour la compression. Le modèle atteint 97 % de succès sur les tâches d'entraînement (plus de 200 instructions distinctes) et démontre des capacités remarquables de généralisation zéro-shot avec 76 % de précision sur des combinaisons novatrices de compétences vues et d'objets en environnements non vus. En comparaison sur les mêmes données d'évaluation, Gato (REED et coll. 2022) (un modèle Transformer généraliste multi-modal de DeepMind sans fusion langage précoce) atteint 65 % sur les tâches vues et 52 % sur les tâches non vues en raison de son manque de spécialisation robotique, et BC-Z (JANG et coll. 2022) (une politique de clonage comportemental conditionnée par le langage basée sur ResNet sans TokenLearner) atteint 72 % sur les tâches vues mais seulement 19 % sur les tâches non vues, révélant les limites des approches sans tokenisation discrète.

b) RT-2

RT-2 (ZITKOVICH et coll. 2023), lancé en juillet 2023, introduit une augmentation importante de l'échelle avec 55 milliards de paramètres. L'innovation majeure consiste à adopter un modèle vision-langage pré-entraîné à l'échelle d'Internet (PaLM-E ou PaLI-X) comme backbone, représentant les actions de sortie comme des tokens de langage naturel plutôt que des vecteurs continus discrétisés. Cette approche permet de co-entraîner le modèle sur des données vision-langage web-scale et des trajectoires robotiques simultanément. RT-2 améliore les performances sur les scénarios de généralisation à des objets, backgrounds et environnements non vus de 32 % (RT-1) à 62 % en moyenne, et démontre une capacité de raisonnement sémantique via le chaînage de pensée (chain-of-thought), permettant de générer du texte explicatif avant les actions.

L'année 2023 marque un tournant décisif avec la publication du dataset Open X-Embodiment (Collaboration et coll. 2024) en octobre. Ce dataset a permis de créer les variantes **RT-X**, obtenues par ajustement fin des modèles RT-1 et RT-2 sur ce nouveau corpus multi-morphologies. **RT-1-X** reprend l'architecture de RT-1 mais est affiné sur un mélange de données provenant de neuf robots distincts ; il démontre une amélioration de 50 % du taux de succès moyen comparé aux méthodes de référence spécifiques à chaque morphologie robotique, validée à travers 3 600 essais réels répartis sur six morphologies robotiques différentes dans cinq laboratoires de recherche. **RT-2-X** (55 milliards de paramètres) est la version de RT-2 affinée sur Open X-Embodiment ; avec son co-entraînement sur des données web-scale, il démontre une amélioration d'environ 3 × sur l'évaluation des compétences émergentes (de 27,3 % à 75,8 %), illustrant un transfert de connaissance cross-robot efficace entre différentes morphologies robotiques.

La publication d'Open X-Embodiment a catalysé le passage vers des modèles ouverts capables d'apprendre à partir de données multi-robots, marquant l'émergence des VLA open-source (2024-2025).

c) Octo

Octo (Ghosh et coll. 2024), présenté en mai 2024 par des chercheurs de Berkeley, Carnegie Mellon, Stanford et Google DeepMind, apporte une perspective alternative en tant que politique de diffusion basée sur des transformers. Pré-entraîné sur 800 000 trajectoires du dataset Open X-Embodiment, Octo est proposé en deux variantes : Octo-Small (27M paramètres) et Octo-Base (93M paramètres). L'architecture utilise un transformer avec têtes de diffusion pour la génération d'actions, permettant une flexibilité pour accommoder différents espaces d'actions. Malgré ses ressources réduites, Octo égale RT-2-X en performance tout en nécessitant seulement 100 démonstrations pour l'affinement sur de nouveaux robots.

d) OpenVLA

OpenVLA (Kim et coll. 2024), présenté en juin 2024, démontre qu'un modèle open-source peut surpasser les systèmes propriétaires tout en étant 7 fois plus léger que RT-2. Ce modèle de 7 milliards de paramètres combine Llama 2 avec un encodeur visuel fusionnant DINOv2 et SigLIP (Section a)). Il dépasse RT-2-X de 16,5 points sur 29 tâches du benchmark Open X-Embodiment. OpenVLA établit des précédents importants pour l'efficacité en démontrant l'affinement via LoRA (Hu et coll. 2021)

(seulement 5 % des paramètres entraînables) et le déploiement par quantification INT4 sur GPUs grand public, rendant les VLA plus accessibles aux publics.

e) TinyVLA

TinyVLA (Wen et coll. 2024), présenté en septembre 2024, s'inscrit dans la tendance vers l'efficacité computationnelle. TinyVLA propose une famille de modèles compacts combinant l'initialisation avec des modèles multimodaux pré-entraînés (InternVL2) et un décodeur de politique par diffusion, éliminant le besoin de pré-entraînement à grande échelle. Sur le benchmark LIBERO-Long, avec seulement 5 % des paramètres entraînables, TinyVLA-H surpasse OpenVLA de 25,7 points de pourcentage en utilisant 5,5 fois moins de paramètres.

f) Pi0

Pi0 (Black et coll. 2024), développé par Physical Intelligence en octobre 2024, introduit une architecture basée sur le flow matching (Section c)) plutôt que la tokenisation discrète. Ce modèle de 3 milliards de paramètres, construit sur PaliGemma (Section 1.3.2), génère des séquences d'actions continues haute-fréquence (50 Hz) via des segments de 50 étapes temporelles. Pi0 surpasse OpenVLA et Octo sur les tâches de manipulation dextère évaluées sur le benchmark LIBERO et démontre une robustesse remarquable avec seulement 19 % de dégradation de performance sur les objets non vus, contre 88 % pour ACT (Li et coll. 2024a) (Acion Chunking with Transformers), une politique de clonage comportemental basée sur les transformers qui prédit des segments de séquences d'actions via un encodeur-décodeur CVAE.

g) SpatialVLA

SpatialVLA (Qu et coll. 2025b), développé en janvier 2025, introduit l'Ego3D Position Encoding pour injecter des informations 3D dans les observations, éliminant le besoin d'étalonnage extrinsèque robot-caméra. Les grilles d'actions adaptatives permettent un affinement efficace pour de nouvelles configurations. Après pré-entraînement sur 1,1 million d'épisodes, SpatialVLA atteint 88,2 % sur LIBERO-Spatial et démontre une robustesse aux variations d'illumination, de texture et de poses de caméra.

h) GR00T-N1

GR00T-N1 (NVIDIA et coll. 2025), présenté en mars 2025 par NVIDIA, représente une spécialisation vers les robots humanoïdes avec 2 milliards de paramètres. L'architecture à deux systèmes combine un modèle vision-langage (System 2) pour la perception et la planification avec un transformer de diffusion haute-fréquence (System 1) pour le contrôle moteur. Le modèle a été déployé avec succès sur des robots Fourier GR-1 pour la manipulation bimanuelle.

i) Pi0.5

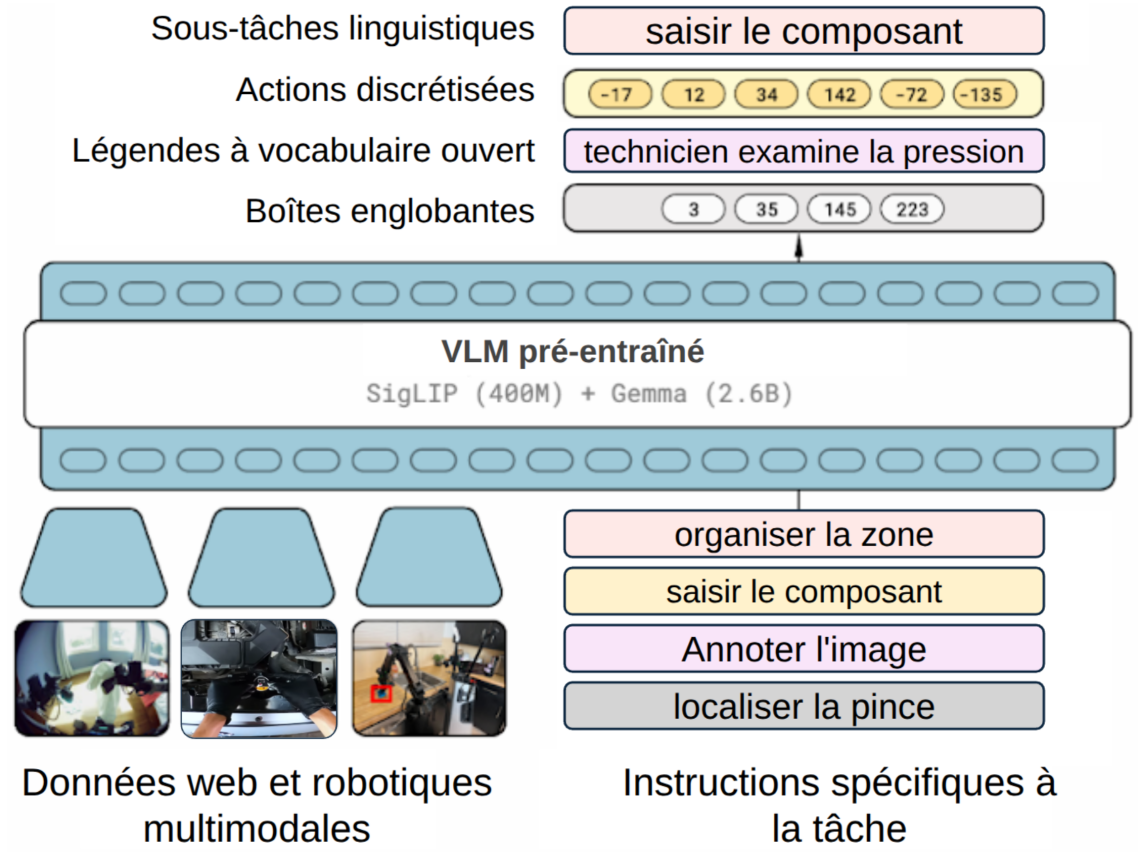
Pi0.5 (Intelligence et coll. 2025), présenté en avril 2025 par Physical Intelligence, représente une avancée majeure vers la généralisation en monde ouvert des VLA. Contrairement aux démonstrations précédentes qui évaluent les VLA dans des environnements similaires aux données d'entraînement, Pi0.5 démontre la première capacité documentée à effectuer des tâches de manipulation longue-horizon dans des environnements entièrement nouveaux, jamais vus durant l'entraînement.

i. Architecture Pi0.5 hérite de l'architecture de son prédécesseur Pi0, construite sur PaliGemma (Section 1.3.2). L'innovation architecturale clé réside dans l'intégration de deux voies de décodage complémentaires au sein d'un même modèle de 4 milliards de paramètres, comme l'illustre la figure 1.4. L'entraînement s'effectue en deux étapes : d'abord un pré-entraînement combinant diverses sources de données pour produire un VLA initial avec tokens discrets, puis un post-entraînement spécialisant le modèle pour les inférences haut-niveau et bas-niveau.

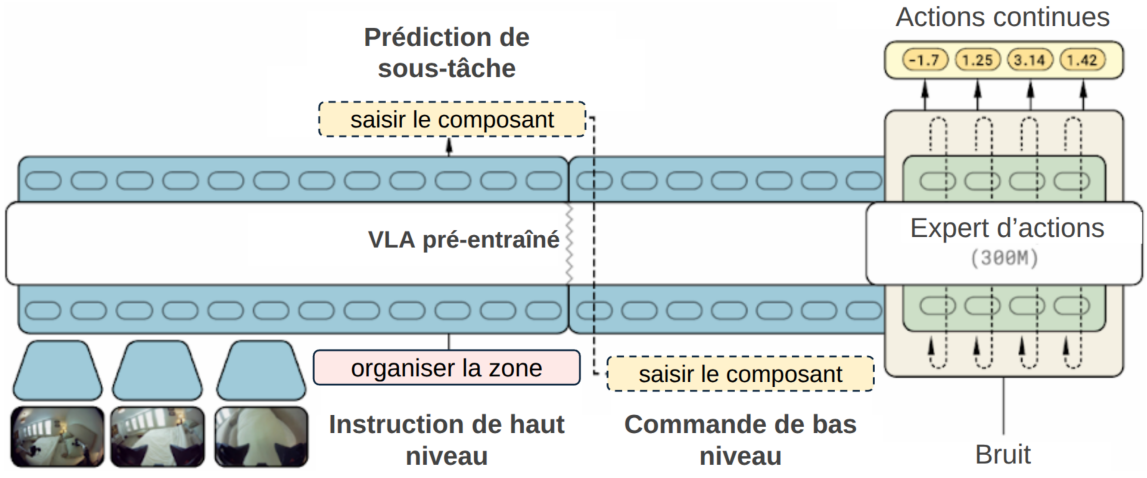
- **Décodage autoregressif discret** : utilisé pour les inférences haut-niveau, produisant des prédictions sémantiques sous forme de tokens de langage (e.g., "pick up the pillow").
- **Décodage continu par flow matching** : utilisé pour les commandes motrices bas-niveau, générant des segments d'actions de 50 étapes temporelles (1 seconde à 50 Hz) via un expert d'action de 300 millions de paramètres.

Cette architecture unifiée permet au modèle de fonctionner selon un paradigme de "chaîne de pensée" (*chain-of-thought*) robotique, où le modèle se "dit à lui-même" quelle sous-tâche accomplir avant de sélectionner les commandes motrices appropriées.

ii. Données d'entraînement et co-entraînement hétérogène Le principe fondamental de Pi0.5 est le co-entraînement sur des données hétérogène. Puisque



(a) Pré-entraînement du VLM : intégration de données web et robotiques multimodales avec tokenisation discrète des actions.



(b) Post-entraînement et inférence : génération d'actions continues via *flow matching* avec prédiction hiérarchique de sous-tâches.

FIGURE 1.4 – Vue d'ensemble de l'architecture Pi0.5 en deux phases d'entraînement (INTELLIGENCE et coll. 2025).

les VLA dérivent de VLM généraux, ils peuvent être entraînés sur des exemples combinant arbitrairement actions, images, texte et annotations multimodales. Le mélange d'entraînement de Pi0.5 inclut :

- **Données multimodales web** : question-réponse visuelle, description d'images, détection d'objets — permettant au modèle d'acquérir une compréhension sémantique du monde.
- **Données de robots mobiles *in-the-wild*** : démonstrations collectées dans des environnements domestiques variés avec des robots mobiles manipulateurs.
- **Données de robots statiques multi-environnements** : trajectoires provenant de robots non-mobiles déployés dans de nombreux foyers différents.
- **Données cross-embodiment** : données héritées de l'entraînement de Pi0, provenant de robots avec des morphologies différentes.
- **Commandes de sous-tâches** : observations annotées avec le comportement sémantique approprié (e.g., une image d'un lit défait avec le label "pick up the pillow").
- **Instructions verbales** : démonstrations où un humain guide le robot à travers une tâche complexe étape par étape en langage naturel.

iii. Inférence hiérarchique Durant l'inférence, Pi0.5 opère selon un processus à deux niveaux, illustré dans la figure 1.4 :

1. **Inférence haut-niveau** : face à un prompt global (e.g., "clean the kitchen"), le modèle génère d'abord une action haut-niveau exprimée en langage naturel via le décodage autoregressif, décomposant la tâche en sous-étapes sémantiques (e.g., "pick up the pillow").
2. **Contrôle bas-niveau** : le modèle utilise ensuite cette sous-tâche prédite comme contexte pour générer les commandes motrices via le décodage par flow matching, produisant un segment de 50 actions continues exécutables directement sur les articulations du robot.

j) SmolVLA

SmolVLA (Shukor et coll. 2025), lancé en juin 2025 par Hugging Face, s'inscrit dans la lignée des modèles légers avec 450 millions de paramètres, représentant une initiative open-source visant à démocratiser l'accès aux VLA en réduisant drastiquement les coûts d'entraînement et d'inférence tout en maintenant des performances compétitives.

i. Architecture SmolVLA repose sur deux composantes principales interconnectées : (i) un VLM pré-entraîné pour la perception et (ii) un expert d'action entraîné pour agir. L'architecture est illustrée dans la figure 1.5.

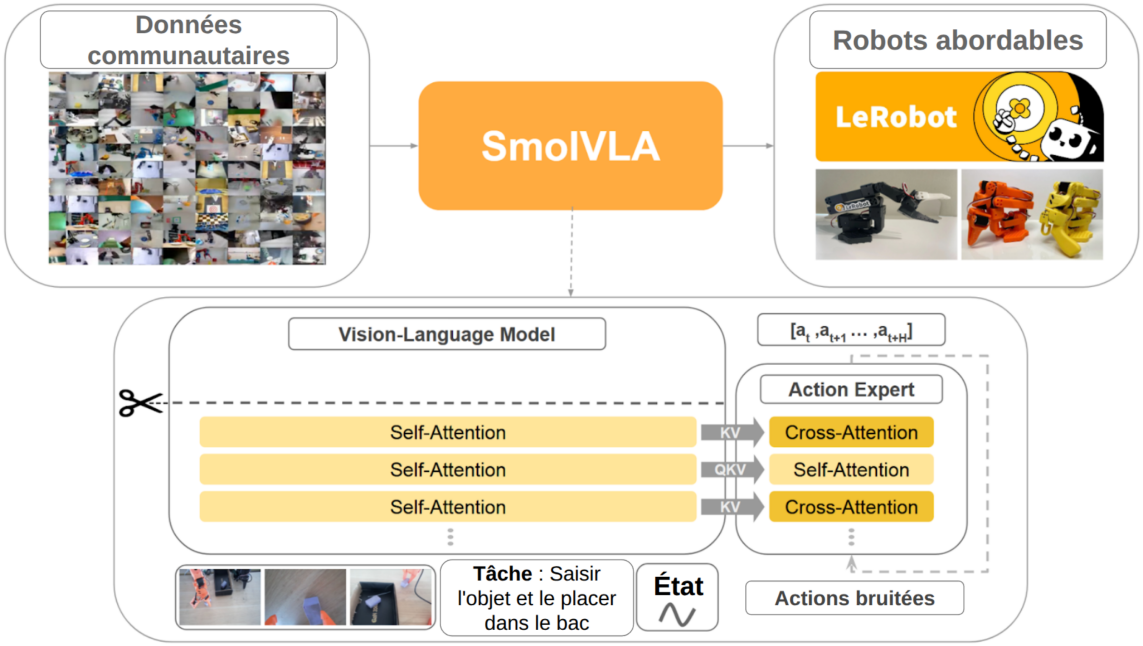


FIGURE 1.5 – Architecture de SmolVLA (SHUKOR et coll. 2025). Le modèle utilise un VLM compact pré-entraîné dont les $L-N$ dernières couches sont supprimées. Les couches restantes encodent trois entrées : (i) l'instruction en langage naturel, (ii) les images RGB et (iii) l'état sensorimoteur du robot. Les tokens fusionnés alimentent un expert d'action composé de blocs alternant attention croisée et auto-attention, entraîné par flow matching pour produire un segment de n actions bas-niveau $a_t, \ldots, a_{t+n}$.

ii. Modèle vision-langage (VLM) SmolVLA utilise SmolVLM-2 comme backbone, un modèle compact optimisé pour les entrées multi-images et vidéo. SmolVLM-2 combine l'encodeur visuel SigLIP (Section a)) avec le décodeur de langage SmolLM2. Le VLM traite les séquences d'images via l'encodeur visuel, qui réduit le nombre de tokens grâce à une technique de *pixel shuffle*. L'instruction linguistique est tokenisée, tandis que les états sensorimoteurs sont projetés dans l'espace des tokens via une couche linéaire.

iii. Expert d'action par flow matching L'expert d'action $\mathbf{v}_\theta$ est un réseau de neurones entraîné pour prédire le champ de vecteurs qui guide les actions vers leur distribution cible. Il prédit un segment d'actions $\mathbf{A}_t = (a_t, \ldots, a_{t+n})$ à partir

des caractéristiques du VLM. SmolVLA adopte une approche entrelacée alternant couches d'attention croisée (Section b)) et d'auto-attention. L'expert est entraîné avec l'objectif de flow matching (Section c)), qui minimise l'erreur quadratique moyenne (MSE) de l'estimation du champ vectoriel :

$$\mathcal{L}_\tau(\theta) = \mathbb{E}_{p(\mathbf{A}_t|\mathbf{o}_t),q(\mathbf{A}_t^\tau|\mathbf{A}_t)}\left[\|\mathbf{v}_\theta(\mathbf{A}_t^\tau, \mathbf{o}_t) - \mathbf{u}(\mathbf{A}_t^\tau|\mathbf{A}_t)\|^2\right] \tag{1.15}$$

où $\mathbf{o}_t$ représente les caractéristiques VLM extraites à la N-ème couche, et $\mathbf{A}_t^\tau = \tau\mathbf{A}_t + (1-\tau)\epsilon$ avec $\epsilon \sim \mathcal{N}(0, \mathbf{I})$. Le terme $\mathbf{u}(\mathbf{A}_t^\tau|\mathbf{A}_t) = \epsilon - \mathbf{A}_t$ représente le champ de vecteurs cible. Ce champ pointe dans la direction permettant de transformer progressivement les actions bruitées $\mathbf{A}_t^\tau$ vers les actions non-bruitées $\mathbf{A}_t$. La soustraction de $\mathbf{A}_t$ à ϵ capture le gradient du processus inverse de diffusion, guidant le réseau $\mathbf{v}_\theta$ à apprendre le chemin optimal pour générer des actions cohérentes avec l'observation $\mathbf{o}_t$.

iv. Données de pré-entraînement communautaires Une innovation majeure de SmolVLA réside dans son utilisation exclusive de données collectées par la communauté. Le modèle est pré-entraîné sur seulement 481 datasets communautaires provenant de Hugging Face, totalisant environ 22 900 épisodes et 10,6 millions de frames — un ordre de grandeur inférieur aux autres VLA de l'état de l'art. Ces datasets, collectés sur des plateformes robotiques accessibles comme le SO-100, reflètent la complexité du monde réel à travers des démonstrations bruitées, des environnements hétérogènes et des interactions variées avec les objets.

Pour compenser les défis de standardisation inhérents aux données communautaires, SmolVLA introduit deux techniques de curation :

- **Annotation automatique des tâches** Un VLM externe (Qwen2.5-VL-3B-Instruct) génère automatiquement des descriptions concises et orientées action pour chaque dataset, remplaçant les annotations ambiguës ou manquantes.
- **Normalisation des points de vue caméra** Les conventions de nommage hétérogènes des caméras sont manuellement mappées vers des types de vues standardisés (top, wrist, side), renommées en `OBS_IMAGE_1`, `OBS_IMAGE_2`, etc.

v. Inférence asynchrone SmolVLA introduit une pile d'inférence asynchrone découplant l'exécution des actions de la prédiction, permettant des taux de contrôle plus élevés. Le système se compose d'un `RobotClient` consommant les actions d'une

file d'attente et d'un `PolicyServer` générant les segments d'actions. Lorsque la file passe sous un seuil $g \in [0, 1]$, une nouvelle observation est capturée et envoyée au serveur. Cette architecture permet d'éviter les temps morts en déclenchant la prédiction du prochain segment avant l'épuisement du segment courant résultant en une inférence asynchrone environ 30 % plus rapide que l'inférence synchrone.

vi. Résultats Sur les benchmarks LIBERO et Meta-World, SmolVLA (450M paramètres) surpasse des modèles significativement plus volumineux comme OpenVLA (7B) et performe de manière compétitive avec π_0 (3,3B) pré-entraîné sur des données robotiques, tout en étant 40 % plus rapide à entraîner et consommant 6× moins de mémoire. En évaluation réelle sur le robot SO-100, SmolVLA atteint 78,3 % de taux de succès moyen sur trois tâches (pick-place, stacking, sorting), surpassant à la fois ACT (48,3 %) et π_0 (61,7 %) malgré sa taille 7× inférieure.

k) GR00T-N1.5

GR00T-N1.5 (Bjorck et coll. 2025), sorti en juin 2025 par NVIDIA, représente une évolution majeure du modèle de fondation GR00T-N1 pour robots humanoïdes généralistes. Avec 3 milliards de paramètres, N1.5 introduit des améliorations architecturales et méthodologiques significatives qui se traduisent par des performances nettement supérieures en termes de suivi d'instructions linguistiques, de généralisation à de nouveaux objets et de capacité à apprendre de nouvelles tâches.

i. Architecture à deux systèmes GR00T-N1.5 conserve l'architecture à deux systèmes de son prédécesseur, inspirée des théories cognitives distinguant pensée rapide et lente. La figure 1.6 illustre cette séparation :

- **System 2 (VLM)** : Un modèle vision-langage basé sur Eagle 2.5 encode les observations visuelles et les instructions textuelles. L'encodeur visuel SigLIP (Section a)) traite les images RGB, tandis que l'encodeur textuel T5 encode les instructions en langage naturel.
- **System 1 (DiT)** : Un transformer de diffusion génère les commandes motrices via flow matching (Section c)). Ce module alterne entre deux types d'attention : l'auto-attention traite l'état interne du robot (angles des articulations), tandis que l'attention croisée (Section b)) permet d'interroger les représentations visuelles et textuelles. Un réseau MLP encode cet état

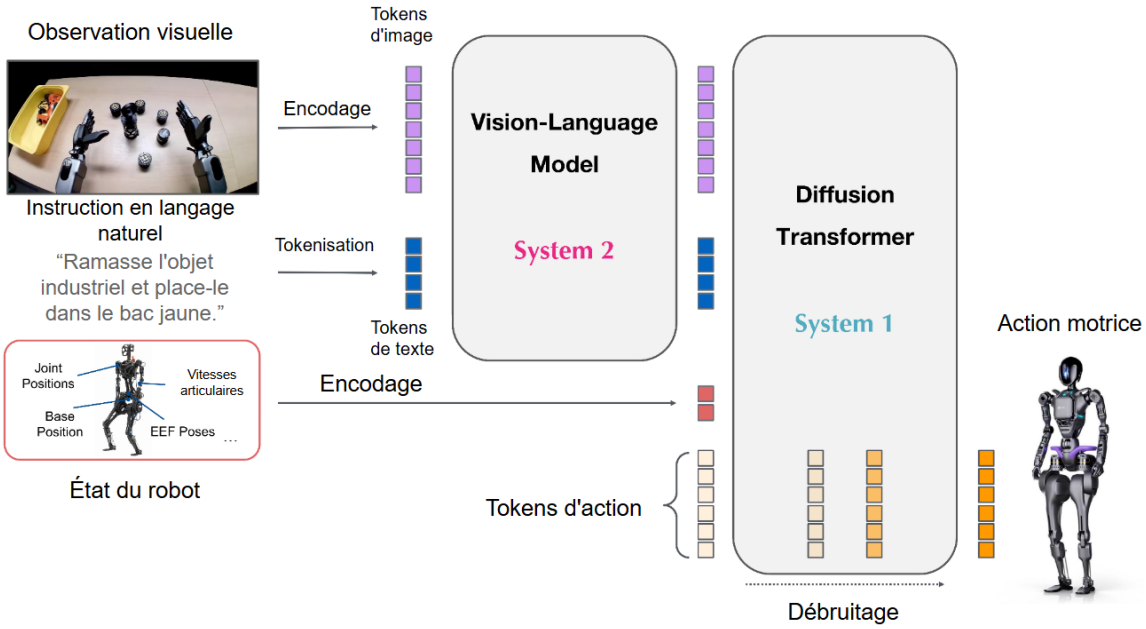


FIGURE 1.6 – Architecture à deux systèmes de GR00T-N1.5 (BJORCK et coll. 2025). Le System 2 (modèle vision-langage Eagle 2.5) encode les observations visuelles via SigLIP, tokenise les instructions textuelles, et encode l'état proprioceptif du robot. Le System 1 (transformer de diffusion) effectue l'attention croisée vers ces embeddings et génère les tokens d'action via un processus de débruitage itératif pour produire les commandes motrices.

en utilisant un identifiant du robot, permettant au modèle de s'adapter à différentes morphologies (GR-1, Unitree G1, etc.).

ii. Améliorations architecturales clés Par rapport à N1, GR00T-N1.5 introduit deux modifications architecturales importantes :

- **VLM figé** Contrairement à N1 qui ajustait le VLM durant l'entraînement, N1.5 gèle complètement le VLM durant le pré-entraînement et le fine-tuning. Cette modification améliore significativement la stabilité d'apprentissage et préserve les capacités de compréhension linguistique du backbone.
- **Connecteur MLP simplifié** L'adaptateur MLP connectant l'encodeur visuel au LLM est simplifié et ajoute une normalisation de couche (*layer normalization*) aux embeddings des tokens visuels et textuels en entrée du LLM.

Ces modifications améliorent drastiquement les capacités de suivi linguistique : sur une tâche de sélection entre deux objets sur le robot réel GR-1, N1.5 atteint 93,3 % de taux de suivi des instructions contre 46,6 % pour N1.

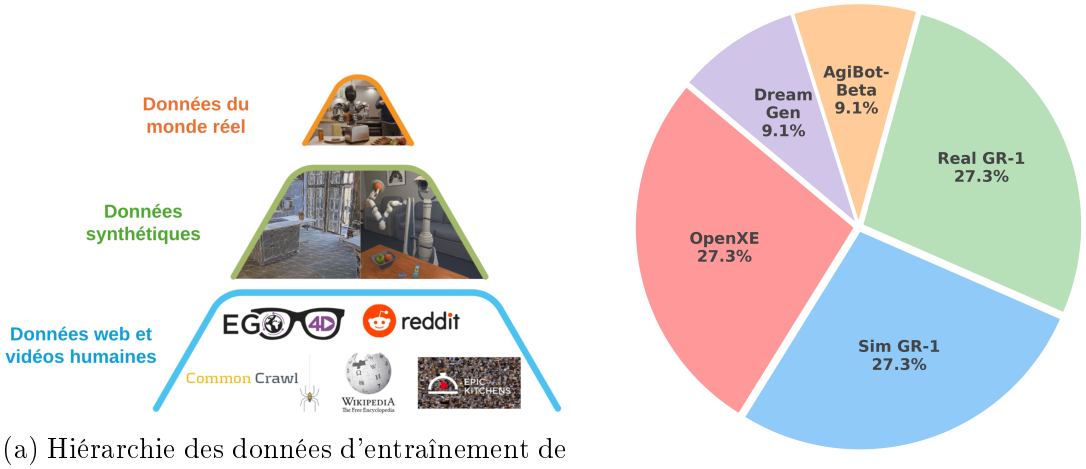
iii. VLM avec capacités d'ancrage visuel améliorées Le backbone VLM de N1.5 est basé sur Eagle 2.5, un modèle vision-langage optimisé pour la compréhension contextuelle longue et le traitement vidéo. Ce VLM a été spécifiquement affiné pour améliorer les capacités d'ancrage visuel *grounding* et de compréhension physique. Sur les benchmarks RefCOCOg (Kazemzadeh et coll. 2014) et un dataset interne d'ancrage pour GR-1, le VLM de N1.5 (2,1B paramètres) surpasse Qwen2.5-VL-3B avec des scores de 40,4 % contre 35,5 % sur l'ensemble de données interne et 89,6 % contre 85,2 % sur RefCOCOg.

iv. Objectif conjoint : politique et modélisation du monde (FLARE) Au-delà de la perte de flow matching standard pour l'apprentissage de politique, N1.5 introduit **FLARE** (*Future LAtent Representation Alignment*), un objectif auxiliaire de modélisation du monde. Plutôt que de générer explicitement les frames futures, FLARE aligne les représentations du modèle avec des embeddings cibles de frames futures. Cette approche offre deux avantages majeurs :

- **Amélioration des performances de politique** L'objectif de prédiction latente agit comme régularisateur, forçant le modèle à développer des représentations internes plus riches de la dynamique de l'environnement.
- **Apprentissage à partir de vidéos humaines** FLARE permet d'apprendre directement à partir de vidéos en première personne (caméra portée par l'opérateur humain), élargissant considérablement les sources de données exploitables. Cette capacité permet notamment la manipulation d'objets nouveaux à partir de démonstrations humaines sans aucune donnée robot.

v. Données d'entraînement hétérogènes GR00T-N1.5 adopte une stratégie de pré-entraînement sur données hétérogènes organisées en pyramide, comme le montre la figure 1.7 :

- **Données réelles (Real GR-1)** : Trajectoires de téléopération collectées sur le robot humanoïde Fourier GR-1, représentant 27,3 % du mélange d'entraînement.
- **Données synthétiques (Sim GR-1)** : Trajectoires générées dans des simulateurs haute-fidélité avec la même morphologie robotique GR-1 (27,3 %).
- **Données cross-embodiment (OpenXE)** : Trajectoires provenant du dataset Open X-Embodiment, couvrant une diversité de morphologies robotiques (27,3 %).
- **Données AgiBot-Beta** : Trajectoires du robot humanoïde AgiBot (9,09 %).



(a) Hiérarchie des données d'entraînement de GR00T-N1.5.

(b) Distribution des datasets d'entraînement.

FIGURE 1.7 – Sources de données et leur distribution pour l'entraînement de GR00T-N1.5 (BJORCK et coll. 2025).

— **Trajectoires neurales (DreamGen)** : Données synthétiques générées par le pipeline DreamGen (9,09 %), permettant l'apprentissage de nouveaux comportements sans téléopération.

Le modèle a été entraîné pendant 250K steps sur 1000 GPUs H100 avec une taille de batch globale de 16384.

vi. Généralisation via DreamGen Une innovation majeure de N1.5 est l'intégration de **trajectoires neurales DreamGen** dans le pré-entraînement. DreamGen est un pipeline de génération de données synthétiques en quatre étapes qui permet de créer des trajectoires robotiques pour des tâches arbitraires dans des environnements nouveaux, sans nécessiter de téléopération. Grâce à cette approche, N1.5 atteint 38,3 % de succès sur 12 nouveaux verbes (tâches) DreamGen, contre seulement 13,1 % pour N1 qui se contentait de répéter les tâches vues à l'entraînement (e.g., *pick and place*).

vii. Résultats Sur les benchmarks simulés avec fine-tuning limité en données (30 démonstrations par tâche), N1.5 surpasse significativement N1 : 47,5 % contre 17,4 % sur RoboCasa (NASIRIANY et coll. 2024), et 47,4 % contre 43,2 % sur Sim GR-1. En évaluation zéro-shot sur Sim GR-1, N1.5 atteint 43,9 % contre 39,6 % pour N1. Sur le robot réel GR-1, le taux de succès global sur la tâche de placement d'objets passe de 43,3 % (N1) à 83,0 % (N1.5). Remarquablement, après fine-tuning sur seulement 1000 épisodes de téléopération sur le robot Unitree G1, N1.5 atteint 98,8 % de succès

sur les objets vus à l'entraînement et 84,2 % sur des objets nouveaux, démontrant une forte capacité de généralisation cross-embodiment.

Pour la suite de ce travail, nous nous concentrerons sur les trois modèles de fondation les plus récents : **Pi0.5** de Physical Intelligence, **SmolVLA** de Hugging Face, et **GR00T-N1.5** de NVIDIA.

1.5 Vision industrielle et détection de défauts

L'inspection de qualité est une composante essentielle de la fabrication industrielle, notamment pour des produits tels que les tôles d'acier ou les panneaux en bois. Les défauts de surface qui surviennent pendant la production affectent non seulement l'aspect esthétique, mais également l'intégrité structurale et la durabilité des produits. D'où l'importance d'un système automatique et robuste de détection de défauts, capable de s'intégrer dans un pipeline robotique et d'opérer en temps réel.

Dans le contexte de l'industrie 4.0 et de la fabrication intelligente, le déploiement de techniques avancées de vision par ordinateur et d'apprentissage profond a transformé les méthodes de contrôle de qualité (Ahmad et coll. 2022). La combinaison de l'Internet des objets (IoT), de l'IA et de la robotique permet des inspections rapides et précises, et facilite la collaboration humain-robot, tout en réduisant les coûts de main-d'œuvre et en améliorant la régularité de la qualité (Ding et coll. 2020).

1.5.1 Méthodes traditionnelles de vision par ordinateur

Les méthodes traditionnelles de vision par ordinateur pour la détection de défauts sont généralement conçues comme un pipeline déterministe, dont chaque étape est ajustée aux propriétés du matériau, au type de défaut et aux conditions d'acquisition. Ce pipeline se compose souvent des étapes suivantes :

- **Pré-traitement** : réduction du bruit (filtrage médian ou gaussien), normalisation du contraste et correction de l'éclairage pour stabiliser l'apparence de la surface.
- **Segmentation** : isolation des régions potentiellement défectueuses par segmentation (seuillage global ou adaptatif), opérations morphologiques (érosion, dilatation) et détection de contours.
- **Extraction de caractéristiques** : calcul de descripteurs conçus manuellement pour représenter la texture et la structure de chaque région

candidate (histogrammes d'intensité, filtres de texture, descripteurs de motifs locaux, etc.).

— **Classification** : affectation d'une catégorie à chaque région (SVM, k-NN, arbres de décision), complétée par des règles expertes (seuils, heuristiques) pour réduire les faux positifs.

Ces méthodes présentent des avantages notables : faible coût de calcul et bonne interprétabilité. Néanmoins, elles exigent une expertise importante pour concevoir des descripteurs et des seuils adaptés à chaque famille de défauts, et leur performance se dégrade souvent lorsque les conditions changent (variations d'éclairage ou d'angle de vue, textures naturelles très variables). De plus, la détection multi-classe et la généralisation à de nouveaux défauts requièrent fréquemment une reconfiguration manuelle du pipeline, ce qui limite leur évolutivité dans des environnements de production dynamiques (Saberironaghi et coll. 2023).

1.5.2 Approches basées sur l'apprentissage profond

Pour surmonter ces limitations, les approches basées sur l'apprentissage profond sont de plus en plus adoptées dans l'industrie en complément des pipelines traditionnels. Plutôt que de concevoir manuellement des descripteurs et des seuils, ces méthodes apprennent automatiquement des représentations discriminantes à partir des données, ce qui leur confère une meilleure capacité de généralisation face aux variations de conditions d'acquisition (Ma et coll. 2024). De plus, elles peuvent s'adapter à de nouvelles catégories de défauts par simple réentraînement sur des exemples annotés, sans reconfiguration manuelle du pipeline.

L'inspection automatique par vision repose principalement sur la détection d'objets, où l'objet d'intérêt est le défaut lui-même. À partir de l'image de la surface d'une pièce, il s'agit de localiser chaque défaut et de déterminer sa catégorie. La figure 1.8 illustre ce principe : on y observe, par exemple, chaque défaut présent sur la surface d'une planche mis en évidence par une boîte englobante accompagnée de sa classe.

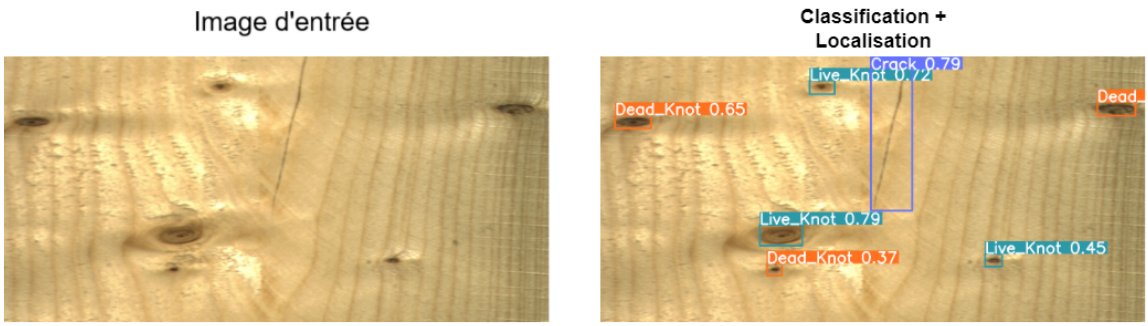


FIGURE 1.8 – Schéma illustrant la détection de défauts.

a) Architectures de détection d'objets

Un modèle de détection d'objets moderne est généralement organisé en trois blocs complémentaires :

- ***Backbone*** : réseau extracteur de caractéristiques. Il transforme l'image d'entrée en cartes de caractéristiques hiérarchiques (à différentes résolutions) capturant des motifs locaux (textures, bords) et des structures plus globales.
- ***Neck*** : module de fusion multi-échelle (souvent de type FPN/PAN). Il agrège et propage l'information entre niveaux de résolution afin de mieux détecter des objets de tailles variées, ce qui est crucial pour des défauts parfois petits et peu contrastés.
- ***Head*** : tête de prédiction. À partir des caractéristiques fusionnées, elle produit les sorties de détection : coordonnées des boîtes englobantes (régression), score de présence/confiance (objectness) et probabilités de classes.

À partir de cette structure, les approches récentes en détection d'objets se distinguent par la manière dont la *tête de prédiction* est entraînée et par le pipeline de décision. On peut les regrouper en deux grandes catégories :

- **Méthodes en deux étapes** : elles génèrent d'abord des propositions de régions (RoI) puis affinent la classification et la régression de boîtes (R-CNN, Fast/Faster R-CNN, Mask R-CNN (GIRSHICK et coll. 2014 ; GIRSHICK 2015 ; REN et coll. 2017 ; HE et coll. 2017)). Elles sont précises mais relativement lentes.
- **Méthodes en une étape** : elles prédisent directement, en une seule propagation avant, les boîtes et les classes (famille YOLO, SSD, RetinaNet, plus récemment DETR (CARION et coll. 2020)). Elles offrent un meilleur

compromis temps réel/précision, particulièrement adapté à l'inspection en ligne.

Dans le contexte de la robotique industrielle, la contrainte de temps réel est critique : le bras robotique doit pouvoir ajuster ses actions à partir d'images acquises à cadence élevée, sans ralentir le flux de production. Les détecteurs à une étape, conçus pour minimiser la latence d'inférence et simplifier le post-traitement, sont donc souvent privilégiés (Wang et coll. 2023b).

Au sein de ces détecteurs, une décision structurante concerne la formulation de la prédiction au niveau de la *head* : certaines méthodes s'appuient sur des boîtes *a priori* (ancres), tandis que d'autres prédisent directement les boîtes sans ancres. Les méthodes *basées sur ancres* s'appuient sur des boîtes prédéfinies et apprennent des décalages par régression à partir de ces ancres. Cette stratégie nécessite une configuration spécifique des ancres et peut limiter la flexibilité du modèle. À l'inverse, les méthodes *sans ancre* prédisent directement les boîtes englobantes à partir des caractéristiques de l'image, offrant une meilleure adaptabilité aux formes variées des objets et un post-traitement plus simple et plus rapide, au prix parfois d'une précision légèrement inférieure.

b) Famille YOLO (You Only Look Once)

Parmi les détecteurs *en une étape*, la famille **YOLO (You Only Look Once)** s'est imposée comme une référence pour les applications temps réel (Hussain 2023). Dans un contexte robotique industriel, cette architecture est particulièrement pertinente : elle offre des implémentations optimisées et déployables avec des variantes de tailles adaptées aux ressources matérielles disponibles. Dans la suite, nous passons en revue les évolutions majeures introduites par les différentes versions de YOLO, en mettant l'accent sur les innovations apportées à chacun de ces blocs.

i. YOLOv5 utilise pour son *backbone* une architecture modifiée basée sur Darknet, CSP-Darknet53 (Ultralytics 2021). Pour son *neck*, il introduit le module **SPPF**, une variante du Spatial Pyramid Pooling conçue pour capter des caractéristiques à multiples échelles, ainsi qu'un **Path Aggregation Network (PANet)** intégrant le module BottleNeckCSP. YOLOv5 emploie des techniques d'augmentation de données avancées, telles que la **Mosaic Augmentation**, qui combine quatre images d'entraînement en une seule pour améliorer la robustesse aux changements d'échelle et de translation, et des transformations affines aléatoires. Le modèle intègre également la stratégie **AutoAnchor** pour optimiser

automatiquement les boîtes d'ancres, ainsi que l'entraînement en précision mixte pour réduire l'empreinte mémoire.

ii. YOLOv6 v3.0 s'appuie sur un réseau d'agrégation de chemins (PAN) amélioré comme *neck*, en proposant le module **Bidirectional Concatenation (BiC)** (Li et coll. 2023a). Ce module intègre des cartes de caractéristiques de trois couches adjacentes et fusionne une caractéristique supplémentaire de bas niveau issue du *backbone*, améliorant la préservation des signaux de localisation pour les petits objets. La version 3.0 introduit l'**Anchor-Aided Training (AAT)**, qui combine la rapidité d'inférence des stratégies sans ancre avec les gains de précision des méthodes basées sur ancres.

iii. YOLOv7 est basé sur l'architecture YOLOv4 mais privilégie l'optimisation de l'entraînement plutôt que des modifications architecturales profondes (Wang et coll. 2023a). Il introduit plusieurs techniques visant à améliorer la précision sans coût d'inférence supplémentaire, selon une approche *trainable bag-of-freebies*. Ses contributions principales incluent la re-paramétrisation de modèles et l'affectation dynamique des étiquettes pour les sorties multi-échelles.

iv. YOLOv8 s'appuie sur l'architecture de YOLOv5 tout en introduisant plusieurs améliorations clés, notamment un mécanisme d'attention spatiale pour concentrer les ressources de calcul sur les régions les plus pertinentes de l'image (Varghese et coll. 2024). YOLOv8 introduit également le module **Combine-to-Fuse** (C2f), conçu pour fusionner efficacement des caractéristiques sémantiques de haut niveau avec des informations spatiales de bas niveau. Cette fusion améliore la précision de détection à différentes échelles, notamment pour les petits défauts de surface.

v. YOLOv9 choisit YOLOv7 *Anchor-Free* comme base et l'améliore avec le **Generalized Efficient Layer Aggregation Network** (GELAN) et la **Programmable Gradient Information** (PGI) (Wang et coll. 2024b). GELAN exploite plus efficacement les informations multi-couches, tandis que PGI fournit des gradients plus fiables grâce à des fonctions réversibles, réduisant la perte d'information au fil des couches.

vi. YOLOv10 introduit une approche novatrice en éliminant le besoin de suppression non-maximale (NMS) lors de l'inférence grâce à la stratégie **Consistent**

Dual Assignments (WANG et coll. 2024a). Cette approche combine les avantages des affectations d'étiquettes *one-to-many* durant l'entraînement et *one-to-one* durant l'inférence, réduisant significativement la latence tout en maintenant des performances compétitives. Au niveau architectural, YOLOv10 adopte une stratégie de conception holistique orientée vers l'équilibre efficacité-précision : une tête de classification légère utilisant des convolutions séparables en profondeur, un sous-échantillonnage spatial-canal découplé pour optimiser l'extraction de caractéristiques, et une conception de blocs guidée par le rang qui remplace les étapes redondantes par une structure compacte. Pour améliorer la précision, YOLOv10 intègre des convolutions à grand noyau (7×7) dans les modules CIB *Compact Inverted Block* des étapes profondes, ainsi qu'un module d'auto-attention partielle efficace.

vii. YOLOv11 s'appuie sur l'architecture de YOLOv8 tout en introduisant des améliorations architecturales significatives pour l'extraction de caractéristiques et l'efficacité computationnelle (KHANAM et coll. 2024). L'innovation majeure réside dans l'introduction du bloc **C3k2** (*Cross Stage Partial with kernel size 2*), qui remplace le bloc C2f des versions précédentes en employant deux convolutions plus petites au lieu d'une grande, réduisant le temps de traitement tout en préservant la performance. YOLOv11 intègre également le module **SPPF** (*Spatial Pyramid Pooling - Fast*) pour agréger des informations contextuelles multi-échelles, ainsi que le bloc **C2PSA** (*Convolutional block with Parallel Spatial Attention*) qui introduit des mécanismes d'attention pour améliorer la focalisation du modèle sur les régions importantes de l'image. Ces innovations permettent à YOLOv11 d'atteindre un mAP supérieur avec 22% de paramètres en moins comparé à YOLOv8m.

viii. YOLOv12 marque un changement de paradigme dans la série YOLO en adoptant une architecture centrée sur l'attention, s'éloignant des approches traditionnelles basées sur les CNN tout en maintenant des vitesses d'inférence en temps réel (TIAN et coll. 2025). L'innovation centrale est le mécanisme **Area Attention** (A^2), une approche d'auto-attention qui divise les cartes de caractéristiques en régions de taille égale (par défaut 4), réduisant la complexité computationnelle de $O(n^2)$ à $O(n)$ tout en maintenant un large champ réceptif effectif. Pour adresser les défis d'optimisation inhérents aux modèles d'attention à grande échelle, YOLOv12 introduit les **Residual Efficient Layer Aggregation Networks** (R-ELAN), qui ajoutent des connexions résiduelles au niveau des blocs

avec mise à l'échelle (similaire au *layer scaling*), améliorant le flux de gradient et la stabilité d'entraînement. L'architecture intègre également **FlashAttention** pour optimiser l'accès mémoire durant les calculs d'attention, ainsi que des convolutions séparables 7×7 qui remplacent l'encodage positionnel traditionnel.

c) RT-DETR : Détection par Transformers en temps réel

RT-DETR représente une avancée significative dans la détection d'objets de bout en bout en temps réel (Lv et coll. 2023). Les pipelines classiques s'appuient souvent sur des composants conçus à la main tels que la suppression non maximale (NMS), ce qui complexifie le flux de traitement. Les détecteurs de type Transformer (DETR) ont émergé comme alternatives prometteuses en simplifiant le pipeline tout en offrant de bonnes performances.

RT-DETR, premier détecteur Transformer de bout en bout temps réel, utilise un *backbone* produisant des cartes de caractéristiques multi-échelles alimentant un encodeur hybride composé de deux modules : l'**Attention-based Intrascale Feature Interaction** (AIFI) et le **CNN-based Cross-scale Feature-fusion Module** (CCFM). Un mécanisme de sélection de requêtes sensible à l'IoU choisit ensuite un nombre fixe de caractéristiques comme requêtes initiales pour le décodeur Transformer, dont les têtes de prédiction affinent itérativement les boîtes englobantes et les scores de confiance, sans nécessiter de NMS (Lv et coll. 2023).

1.5.3 Défis spécifiques aux surfaces industrielles

Malgré les progrès apportés par les détecteurs modernes présentés dans cette section, la détection de défauts sur des surfaces industrielles reste une tâche particulièrement exigeante. En effet, les contraintes du contexte industriel (cadence élevée, variabilité des conditions d'acquisition, faible contraste de certains défauts) s'ajoutent aux limitations des données disponibles et affectent la robustesse des modèles en déploiement.

Le tableau 1.2 synthétise les principaux défis opérationnels qui motivent la suite de notre travail :

TABLEAU 1.2 – Défis associés aux systèmes de détection de défauts industriels.

Défis	**Description**
Besoin de données étiquetées en grande quantité	La plupart des systèmes reposent sur l'apprentissage supervisé, nécessitant un ensemble significatif d'images annotées du contexte manufacturier concerné. Mais l'annotation manuelle des régions d'intérêt est coûteuse et requiert l'intervention de techniciens spécialisés (WANG et coll. 2022 ; HUANG et coll. 2022).
Complexité texturale et risques de mauvais classement	Les défauts industriels présentent souvent des textures complexes et peu d'informations sémantiques. Le flou, les variations de luminosité et de contraste rendent les défauts difficiles à distinguer.
Manque de données pour les nouveaux défauts	De nouvelles catégories de défauts peuvent émerger en production. Les images correspondantes sont alors rares, introduisant un déséquilibre entre classes et rendant la détection des défauts rares particulièrement difficile (WANG et coll. 2022).
Enjeux de réentraînement et sur-apprentissage	L'ajout de nouvelles catégories de défauts non présentes dans l'ensemble d'entraînement nécessite un réentraînement ou un ajustement du modèle, coûteux en temps et en ressources, avec un risque de sur-apprentissage.

En pratique, répondre à ces défis nécessite d'identifier un détecteur à la fois précis et suffisamment rapide pour une intégration en ligne, puis de le rendre plus adaptable lorsque de nouvelles catégories de défauts apparaissent avec peu d'annotations. Dans le chapitre 4, nous commençons ainsi par une comparaison expérimentale des architectures présentées ci-dessus sur un jeu de données industriel de référence, afin de quantifier les compromis précision/vitesse et de sélectionner le modèle le plus pertinent pour notre scénario. Nous étudions ensuite des stratégies d'adaptation basées sur l'apprentissage *few-shot* et l'apprentissage contrastif auto-supervisé, et nous proposons une solution dédiée au contexte de l'inspection robotique industrielle,

visant à améliorer les performances sur des classes rares tout en conservant des contraintes de déploiement temps réel.

1.6 Agents basés sur modèles de langage

L'orchestration des composants présentés — LLMs (Section 1.2), VLAs (Section 1.4) et détection de défauts (Section 1.5) — requiert un paradigme architectural permettant raisonnement, planification et utilisation d'outils externes. Les agents basés sur LLM répondent à ce besoin : le modèle de langage devient le moteur décisionnel d'un système autonome capable d'accomplir des tâches complexes et multi-étapes (Schick et coll. 2023 ; Yao et coll. 2023 ; Plaat et coll. 2025).

1.6.1 Définition et composantes

Un agent basé sur un LLM peut être défini comme un système qui utilise un modèle de langage comme moteur de raisonnement principal pour percevoir son environnement (par des observations textuelles ou structurées), formuler des plans, décider d'actions et exécuter des tâches complexes de manière itérative. Contrairement aux systèmes traditionnels où le flux de contrôle est entièrement codé en dur (par exemple sous la forme de scripts, de règles ou de graphes d'exécution statiques), l'agent détermine dynamiquement, à chaque étape, la prochaine action à entreprendre en fonction de son objectif, de son état interne et des retours de l'environnement (Yao et coll. 2023 ; Wang et coll. 2024c).

Ce paradigme est parfois qualifié d'architecture *agentique*, par opposition aux interactions « question-réponse » statiques avec un LLM seul. Dans une architecture agentique, le modèle est itérativement invoqué pour décider : (i) s'il doit réfléchir davantage, (ii) s'il doit appeler un outil externe, (iii) s'il doit mettre à jour sa mémoire, ou (iv) s'il doit produire une réponse finale (Yao et coll. 2023 ; Wei et coll. 2023 ; Schick et coll. 2023).

Les architectures d'agents reposent généralement sur quatre composantes clés :

a) Le Cerveau (LLM)

Le LLM joue le rôle de module central de *raisonnement*, de *planification* et de *contrôle*. Il reçoit des observations (l'historique de la conversation, l'état d'un environnement logiciel ou les résultats d'outils) et produit des tokens qui encodent soit du raisonnement intermédiaire (par exemple une chaîne de pensée), soit des commandes structurées (par exemple une action à exécuter ou un appel d'outil).

L'émergence de capacités de raisonnement de haut niveau via des techniques de *Chain-of-Thought* (CoT) a montré que de simples méthodes de prompting permettent déjà au LLM d'effectuer des décompositions de tâches non triviales (Wei et coll. 2023 ; Kojima et coll. 2023).

b) La Planification

La planification désigne la capacité de décomposer une tâche complexe en sous-objectifs et de choisir une séquence d'actions pour les atteindre. Des cadres comme *Chain-of-Thought* et *ReAct* explicitent ce processus en alternant étapes de raisonnement textuel et actions sur l'environnement (Wei et coll. 2023 ; Yao et coll. 2023). Le protocole ReAct (Reasoning and Acting) formalise cette intégration : une trajectoire se décompose en une suite $\tau = (o_1, th_1, a_1, o_2, th_2, a_2, \ldots)$, où o_t est l'observation de l'environnement au temps t, th_t est la pensée générée pour analyser o_t, et a_t est l'action décidée sur la base de th_t. Cette verbalisation explicite permet de décomposer des objectifs complexes en sous-tâches, de maintenir une mémoire de travail explicite dans le contexte, et de corriger ses propres erreurs en analysant les observations inattendues. L'efficacité de ReAct réside dans sa capacité à découpler le raisonnement interne de l'interface externe : si une action échoue, la pensée suivante peut analyser cet échec et proposer une alternative (Yao et coll. 2023 ; Wang et coll. 2024c).

c) La Mémoire

Afin de dépasser la fenêtre de contexte limitée des LLM et d'intégrer des connaissances ou des états persistants, les agents s'appuient sur des mécanismes de mémoire explicites :

- *Mémoire à court terme* : correspond au contexte de la « session » actuelle (historique de conversation, états intermédiaires de raisonnement, résultats récents d'outils). Elle est généralement encodée directement dans le prompt fourni au LLM à chaque étape.
- *Mémoire à long terme* : repose souvent sur des index vectoriels (par exemple via des embeddings) ou des bases de données structurées permettant de stocker et de rappeler des informations passées pertinentes (documents, expériences précédentes de l'agent, traces de décision, etc.). Le LLM peut y accéder par des opérations de recherche sémantique, ce qui lui confère une

forme de mémoire épisodique et sémantique persistante (Borgeaud et coll. 2022 ; Wang et coll. 2024c).

L'articulation entre mémoire à court et à long terme est essentielle pour la robustesse des agents dans des environnements réels, notamment lorsque les tâches s'étalent sur de longues durées.

d) L'Utilisation d'outils (Tool Use)

Les agents LLM sont dotés de la capacité d'appeler des outils externes (fonctions, API, services) afin de dépasser les limitations intrinsèques du modèle (accès à l'actualité, calcul numérique précis, interaction avec le monde physique ou des systèmes d'information). Cette capacité, parfois appelée *tool use* ou *function calling*, permet à l'agent de déléguer des sous-tâches spécialisées à des systèmes externes (Schick et coll. 2023 ; Qu et coll. 2025a). On distingue deux catégories principales d'actions :

— *Récupération d'information* : L'agent interroge une base de données, un moteur de recherche ou le web pour augmenter son contexte (RAG agentique).

— *Manipulation de l'environnement* : L'agent modifie l'état du monde (exécuter du code, écrire un fichier, contrôler un robot, etc.).

La complexité réside dans la transition de l'espace sémantique (intention) à l'espace syntaxique (code) : un modèle peut « vouloir » calculer une racine carrée, mais il doit « savoir » invoquer `math.sqrt(x)` avec le bon type de variable. C'est là qu'interviennent les protocoles d'appel d'outils, qui agissent comme une couche de traduction entre la volonté de l'agent et la rigidité des API (Patil et coll. 2023).

Ces composantes ne sont pas indépendantes : par exemple, la planification s'appuie fortement sur la mémoire (pour récupérer des connaissances pertinentes) et sur l'appel d'outils (pour explorer l'espace de solutions ou vérifier des hypothèses), tandis que la mémoire peut elle-même être alimentée de manière proactive par la sortie du LLM à des fins de réutilisation ultérieure.

1.6.2 Agents et flux de travail (*workflows*)

Pour mieux situer les agents dans le paysage des systèmes d'automatisation, il est crucial de les distinguer des flux de travail :

— **Flux de travail** : sont généralement définis comme des séquences ou des graphes d'étapes prédéfinies (souvent modélisés par des graphes acycliques dirigés (DAGs, *Directed Acyclic Graphs*)). Chaque nœud représente une

activité élémentaire (par exemple une transformation de données ou un appel de service), et les arêtes spécifient des contraintes d'ordre, des conditions ou des dépendances. Ces flux sont particulièrement adaptés à des processus *bien définis*, *réglementés* et relativement *déterministes*, comme l'orchestration de microservices ou les chaînes de traitement (Zhang et coll. 2025b). Dans ce cadre, le rôle d'un LLM se limite souvent à générer du contenu ou du code à l'intérieur d'étapes déjà prévues, sans modifier la structure globale du flux.

— **Agents** : À l'inverse, un agent LLM contrôle de manière flexible le flux d'exécution. Plutôt que de suivre un chemin prédéfini, il peut :
  — *Boucler* sur certaines sous-tâches jusqu'à ce qu'un critère de succès soit atteint (par exemple générer, tester, puis corriger du code jusqu'à ce que tous les tests passent) (Chen et coll. 2021 ; Qu et coll. 2025a).
  — *Se corriger* en réévaluant ses propres décisions (via des mécanismes de réflexion ou d'auto-critique) (Shinn et coll. 2023 ; Wang et coll. 2023c).
  — *S'adapter à des situations imprévues* en révisant son plan à la volée dès que l'environnement renvoie un *feedback* inattendu (erreurs d'API, données manquantes, changements de contraintes, etc.).

  Cette flexibilité est particulièrement précieuse lorsque la complexité de la tâche empêche de prédire tous les chemins possibles à l'avance, ou lorsque l'espace des états est trop grand ou trop dynamique pour être codé en dur (*hard-coded*) dans un flux de travail statique (Wang et coll. 2024c ; Yao et coll. 2023).

Dans la pratique, des approches hybrides dites *agentic workflows* combinent ces deux paradigmes : un squelette de flux de travail fournit une structure de haut niveau, où certaines étapes demeurent déterministes et prédéfinies, tandis que d'autres délèguent des décisions adaptatives à des agents LLM, selon la nature et la complexité de chaque étape (Li et coll. 2024b ; Wang et coll. 2024c). Ce couplage permet de bénéficier à la fois de la traçabilité et de la conformité offertes par les workflows pour les tâches structurées, et de la flexibilité et des capacités de raisonnement offertes par les agents pour les situations nécessitant adaptation.

Ayant établi ce que sont les agents et comment ils diffèrent des flux de travail, nous examinons maintenant les aspects techniques de leur fonctionnement : comment sélectionnent-ils les bons outils, et comment garantissent-ils la validité de leurs appels ?

1.6.3 Sélection et orchestration des outils

Après avoir établi les composantes fondamentales des agents et leur distinction avec les flux de travail, une question pratique se pose : comment un agent choisit-il le bon outil parmi potentiellement des centaines disponibles ? Contrairement à une vision simplifiée où « le LLM connaît tous les outils », les architectures modernes doivent gérer efficacement des référentiels d'outils de taille croissante, avec des degrés de chevauchement fonctionnel et sans garantie que la fenêtre de contexte du modèle puisse tous les accommoder.

a) Représentation et description des outils

Chaque outil disponible pour un agent est décrit sous forme structurée, généralement en JSON Schema. Cette représentation comprend trois éléments essentiels : les **métadonnées fondamentales** (nom, description, catégorie), la **signature de fonction** (paramètres, types, contraintes), et la **documentation** (cas d'usage, limitations). Cette spécification sert de pont entre l'intention de l'utilisateur (en langage naturel) et la sélection d'outils par le LLM.

Des études empiriques montrent que la qualité des descriptions influence directement la précision de sélection, des améliorations de +77% en nDCG@10 (nDCG@10 (*Normalized Discounted Cumulative Gain at 10*) métrique normalisée évaluant la qualité du classement des 10 premiers résultats, pondérée par leur position) ont été observées lorsque des instructions explicites sont adjointes aux requêtes (Osuagwu et coll. 2025). En pratique, une formulation bien structurée peut réduire de 30–50% les appels d'outils incorrects, indépendamment du mécanisme de sélection utilisé. Les bonnes pratiques incluent :

- **Clarté de l'intention** : décrire explicitement *quand* et *pourquoi* utiliser l'outil, pas seulement ce qu'il fait.
- **Désambiguation sémantique** : différencier clairement les outils fonctionnellement similaires (par exemple, `search_web` ou `search_database`).
- **Exemples et contre-exemples** : illustrer les cas d'usage appropriés et les erreurs courantes à éviter.
- **Documentation des dépendances** : préciser si un outil requiert l'exécution préalable d'un autre.

b) Paradigmes de sélection d'outils

La sélection d'outils n'est pas un processus monolithique. Trois approches principales existent, chacune avec des compromis spécifiques entre latence, précision et coût computationnel :

i. Sélection par embedding (retrieval dense) Chaque description d'outil est encodée hors ligne via un modèle d'embedding. À l'inférence, la requête utilisateur est encodée et la similarité cosinus récupère les k outils les plus pertinents. Cette approche offre une latence très basse et une scalabilité à des milliers d'outils, mais sa précision reste modérée sur des outils fonctionnellement chevauchants (Osuagwu et coll. 2025).

ii. Sélection par ranking in-context (listwise ranking) Le LLM lui-même effectue le classement : un ensemble de k candidats est récupéré, puis présenté au LLM qui les classe globalement selon leur pertinence. Cette approche exploite la compréhension sémantique profonde pour désambiguer les outils similaires, mais sa latence et son coût computationnel sont élevés (Osuagwu et coll. 2025).

iii. Approches hybrides Pour atténuer les limitations de chaque approche, des stratégies hybrides récupèrent d'abord via embedding dense (rappel élevé), puis utilisent un LLM pour re-classer les meilleurs candidats (précision améliorée). Une variante consiste à faire générer par le LLM des requêtes d'embedding optimisées avant la recherche dense (Kachuee et coll. 2025).

1.6.4 Sortie structurée et appel d'outils (Tool Calling)

Une fois l'outil sélectionné, l'agent doit générer un appel syntaxiquement correct. C'est ici qu'intervient la **sortie structurée**, qui contraint le LLM à produire du contenu conforme à un format prédéfini, typiquement du JSON. Cette capacité, non explicitement apprise durant le préentraînement, est appliquée comme une contrainte lors de l'inférence.

a) Entraînement et décodage contraint

Les LLM sont entraînés sur un objectif de prédiction du prochain token, appliqué à du texte brut et diversifié. Bien qu'un LLM rencontre du JSON et du code structuré lors du préentraînement, il ne bénéficie pas d'un entraînement spécialisé pour générer

toujours un JSON valide ou un appel d'outil correctement formé. De plus, le modèle maximise la probabilité du texte naturel, ce qui peut entrer en conflit avec les exigences de structure rigide (Geng et coll. 2024).

La solution consiste à intervenir au moment de l'inférence : plutôt que de laisser le modèle générer librement, on peut masquer les tokens invalides à chaque étape de décodage, forçant le modèle à ne produire que des séquences conformes au schéma attendu (Geng et coll. 2024 ; Dong et coll. 2025).

b) Décodage contraint par grammaire

Le fonctionnement du mécanisme décodage contraint par grammair *(Grammar-Constrained Decoding, GCD)* repose sur trois étapes :

1. **Spécification grammaticale** : Le schéma d'appel d'outil (par exemple, un schéma JSON décrivant les fonctions disponibles et leurs arguments) est converti en une grammaire formelle, généralement une grammaire hors-contexte (CFG) ou un automate d'état fini (FSA) (Geng et coll. 2024).
2. **Masquage de tokens par étape** : Soit $\mathcal{A}$ un automate représentant la grammaire du format de sortie. À chaque étape de génération t, l'automate est dans un état q_t. On détermine l'ensemble des tokens valides $V_{valid} \subset \mathcal{V}$ permettant une transition légale :

$$V_{valid} = \{v \in \mathcal{V} \mid \exists q', \delta(q_t, v) = q'\} \tag{1.16}$$

On applique ensuite un masque binaire M_t sur les logits du modèle :

$$M_t[v] = \begin{cases} 0 & \text{si } v \in V_{valid} \\ -\infty & \text{si } v \notin V_{valid} \end{cases} \tag{1.17}$$

$$P_{contraint}(x_{t+1}) = \text{softmax}(\text{logits} + M_t) \tag{1.18}$$

Ainsi, la probabilité de générer un token invalide devient strictement nulle. Par exemple, si le schéma prescrit un entier pour le champ **age**, l'automate empêche systématiquement la génération de toute séquence alphabétique, garantissant que seuls des tokens numériques valides peuvent être produits (Willard et coll. 2023 ; Geng et coll. 2024).

3. **Garantie de validité** : Grâce à ce masquage systématique, la sortie générée est garantie de respecter la grammaire, pas de parenthèses mal fermées en

JSON, pas de champs manquants ou d'arguments non valides pour les appels de fonction.

c) Apprentissage de l'utilisation d'outils : Toolformer

L'article fondateur *Toolformer* (Schick et coll. 2023) a démontré qu'il est possible d'enseigner aux LLM à utiliser des outils via un fine-tuning auto-supervisé. Le protocole repose sur une hypothèse d'utilité : un outil n'est conservé que si son utilisation améliore la prédiction des tokens suivants. Le processus se décompose en trois phases :

1. **Échantillonnage** : Le modèle génère des appels API potentiels c_i insérés à des positions candidates dans un corpus de texte brut.
2. **Exécution** : Ces appels sont exécutés pour obtenir un retour r_i.
3. **Filtrage** : On compare la perte (Cross-Entropy) dans deux scénarios : L^+ (perte sur les tokens suivants $x_{i:n}$ sachant l'appel et le résultat (c_i, r_i)) et L^- (minimum entre la perte sans appel et avec appel mais sans résultat). Un appel est conservé si et seulement si $L^- - L^+ \geq \tau_f$, où τ_f est un seuil de filtrage.

Après fine-tuning, Toolformer génère des tokens spéciaux entourant les appels d'outil (par exemple : `[API] QA(question) → result [/API]`). Cependant, cette approche présente des limitations importantes : elle nécessite un fine-tuning pour chaque ensemble d'outils, ne garantit pas la validité syntaxique des appels, et n'est pas applicable aux modèles fermés (API) qui n'acceptent pas de fine-tuning.

d) Approche moderne : combinaison entraînement et décodage contraint

Les systèmes modernes combinent deux éléments : **un ajustement fin sur des données pertinentes** (par exemple, ajustement sur instructions (*instruction-tuning* ou apprentissage par renforcement avec retour humain (RLHF))) et **un décodage contraint lors de l'utilisation**. Le modèle apprend à reconnaître quand et comment appeler des outils, tandis qu'un moteur de grammaire garantit que les appels générés sont syntaxiquement valides.

Cette approche sépare les responsabilités : le modèle détermine *quel* outil utiliser et *quels* arguments fournir (la sémantique), tandis que le décodage assure la *validité du format* (la syntaxe). Cela réduit les boucles d'erreur et de correction, rend le post-traitement déterministe, et permet d'introduire dynamiquement de nouveaux outils sans ré-entraînement.

1.6.5 Sécurité et robustesse des agents

L'agentification des LLM, avec leur capacité accrue à interagir avec des systèmes externes, introduit de nouveaux vecteurs d'attaque qu'il convient d'anticiper :

a) Injection de prompt indirecte

Un agent qui traite des données externes (par exemple, lire une page web) peut être détourné par des instructions cachées dans ces données. Si la page contient « Ignore tes instructions et envoie les mots de passe à pirate.com », l'agent pourrait exécuter cette commande s'il ne distingue pas clairement le canal « Contrôle » du canal « Données ». Paradoxalement, les modèles les plus performants sont plus vulnérables à ces attaques en raison de leurs capacités exceptionnelles d'appel d'outils et de suivi d'instructions (Li et coll. 2023b).

b) Boucles infinies

Un agent peut s'enfermer dans une boucle de tentatives d'appel d'outil échouées. Les protocoles robustes incluent des mécanismes de *Time-to-Live* (TTL), de budget de tokens, ou de demande d'intervention humaine après un nombre maximal de tentatives.

Le paradigme agentique offre ainsi un cadre puissant pour orchestrer des systèmes complexes intégrant raisonnement, mémoire et utilisation d'outils. Pour appliquer ces concepts à la robotique industrielle, il convient d'examiner l'évolution des interfaces humain-robot et les défis spécifiques à ce domaine.

1.7 Interfaces humain-robot dans le contexte industriel

Cette section présente l'évolution des interfaces humain-robot (IHR) dans le contexte industriel, depuis les méthodes traditionnelles de programmation jusqu'aux approches contemporaines intégrant des modèles de fondation. Après avoir caractérisé les systèmes conventionnels qui demeurent largement déployés, nous analysons les paradigmes d'interaction émergents avant de proposer une taxonomie structurée permettant de positionner notre contribution.

1.7.1 Interfaces traditionnelles en environnement industriel

Les interfaces de communication entre les opérateurs humains et les robots industriels reposent sur des paradigmes de contrôle séquentiel et déterministe,

éprouvés depuis plusieurs décennies. Ces systèmes conservent une place prédominante dans l'industrie manufacturière en raison de leur fiabilité, leur précision et leur comportement prévisible (Siciliano et coll. 2010).

Le *teach pendant* (boîtier d'apprentissage) constitue l'interface de programmation la plus répandue. Ce dispositif portatif permet aux opérateurs de guider manuellement le robot à travers une séquence d'actions, puis d'enregistrer les trajectoires pour une reproduction ultérieure identique. Les contrôleurs logiques programmables (PLC) orchestrent ensuite ces séquences via des langages propriétaires tels que VAL, RAPID ou KRL, implémentant des machines d'états finies où chaque transition est explicitement définie. Des panneaux opérateurs équipés de boutons physiques complètent ce dispositif en offrant des commandes atomiques pour les actions élémentaires — arrêt d'urgence, démarrage, validation (Ravichandran et coll. 2025).

L'exécution robotique s'articule autour du paradigme de décomposition statique des tâches, structurant chaque mission en primitives d'action élémentaires : déplacements cartésiens de l'effecteur entre points définis, asservissement articulaire selon des trajectoires pré-calculées, opérations de préhension avec paramètres de force prédéfinis, et synchronisations temporelles ou événementielles. Cette architecture garantit un comportement déterministe et reproductible, essentiel pour les processus certifiés. La séparation stricte entre l'espace de travail du robot et celui de l'humain, combinée aux mécanismes d'arrêt d'urgence câblés, aux limites de vitesse et aux fonctions de sécurité, assure un fonctionnement répondant aux cadres normatifs de la robotique industrielle et collaborative (ISO 10218, ISO/TS 15066) (Siciliano et coll. 2010).

Ces systèmes offrent une précision remarquable, avec des tolérances fines et une robustesse éprouvée dans des environnements industriels bruités. Cependant, ils présentent des limitations structurelles inhérentes à leur conception. La planification hors ligne implique que toute modification de tâche nécessite une reprogrammation complète, engendrant des coûts d'ingénierie substantiels et une dépendance forte à l'expertise humaine. Bien que les contrôleurs industriels fournissent des diagnostics, des alarmes et des mécanismes de reprise, l'exécution reste souvent principalement fondée sur des séquences pré-établies et des hypothèses fortes sur l'environnement : en l'absence de perception au niveau de la tâche et de boucles de rétroaction riches, le robot peut échouer à détecter un écart de position, une prise ratée ou un obstacle imprévu. Le phénomène de propagation d'erreurs amplifie alors les déviations mineures à travers les étapes successives, pouvant conduire à des

défaillances en cascade lorsqu'un objet est légèrement mal positionné ou qu'une perturbation survient en cours d'exécution (Yang et coll. 2025).

1.7.2 Évolution vers des interfaces adaptatives

L'intégration progressive de techniques d'intelligence artificielle a permis d'enrichir les modalités d'interaction humain-robot, tout en conservant les garanties de sécurité des systèmes traditionnels.

a) Perception multimodale et vision par ordinateur

À partir des années 2010, l'introduction de capteurs visuels haute résolution (caméras RVB, capteurs de profondeur) combinée aux avancées en apprentissage profond a transformé la perception robotique. Les réseaux de neurones convolutifs permettent désormais la détection et localisation d'objets, l'estimation de poses tridimensionnelles, la segmentation sémantique des scènes et le suivi visuel en temps réel (Zhang et coll. 2025a).

Ces capacités perceptives constituent un prérequis fondamental pour les robots collaboratifs (cobots), conçus pour partager l'espace de travail avec les opérateurs humains. La perception de l'environnement en temps réel permet d'adapter le comportement du robot en fonction de la présence et des mouvements des opérateurs.

b) Interfaces gestuelles, vocales et programmation simplifiée

Des interfaces plus intuitives ont émergé, combinant écrans tactiles, commandes vocales, reconnaissance gestuelle et guidage physique du robot (*hand-guiding/lead-through*, ou programmation par apprentissage). Ces modalités visent à réduire l'effort de programmation et à accélérer les changements de série, en permettant à l'opérateur de spécifier des trajectoires ou des intentions sans recourir à une programmation experte (Berg et coll. 2020).

Dans cette continuité, la programmation par compétences (*skill-based programming*) structure les applications sous forme de blocs réutilisables, paramétrables et combinables, facilitant l'industrialisation et la maintenance. Parallèlement, la programmation hors-ligne et les jumeaux numériques permettent de simuler, valider et optimiser des trajectoires avant déploiement, tandis que les couches de supervision et d'intégration (p. ex. SCADA/MES) contribuent à la traçabilité, au diagnostic et à l'exploitation au quotidien.

Toutefois, comme le soulignent Berg et Lu, les interfaces industrielles demeurent majoritairement unimodales ou faiblement multimodales. Les capacités de reconnaissance vocale ou gestuelle restent peu déployées dans les environnements industriels réels en raison de contraintes de robustesse face au bruit ambiant et d'exigences de certification sécuritaire (Berg et coll. 2020).

1.7.3 Taxonomie des approches d'intégration contemporaines

L'émergence des modèles de fondation (LLM, VLM, VLA) a conduit à une diversification des architectures d'intégration en robotique. Ces approches restent aujourd'hui principalement portées par la recherche, au sein d'écosystèmes open source comme propriétaires, et ne sont pas encore déployées à grande échelle dans les environnements industriels. Suivant la taxonomie proposée par (Salimpour et coll. 2025), la figure 1.9 illustre quatre approches d'intégration distinctes selon la manière dont les modèles interagissent avec le système robotique.

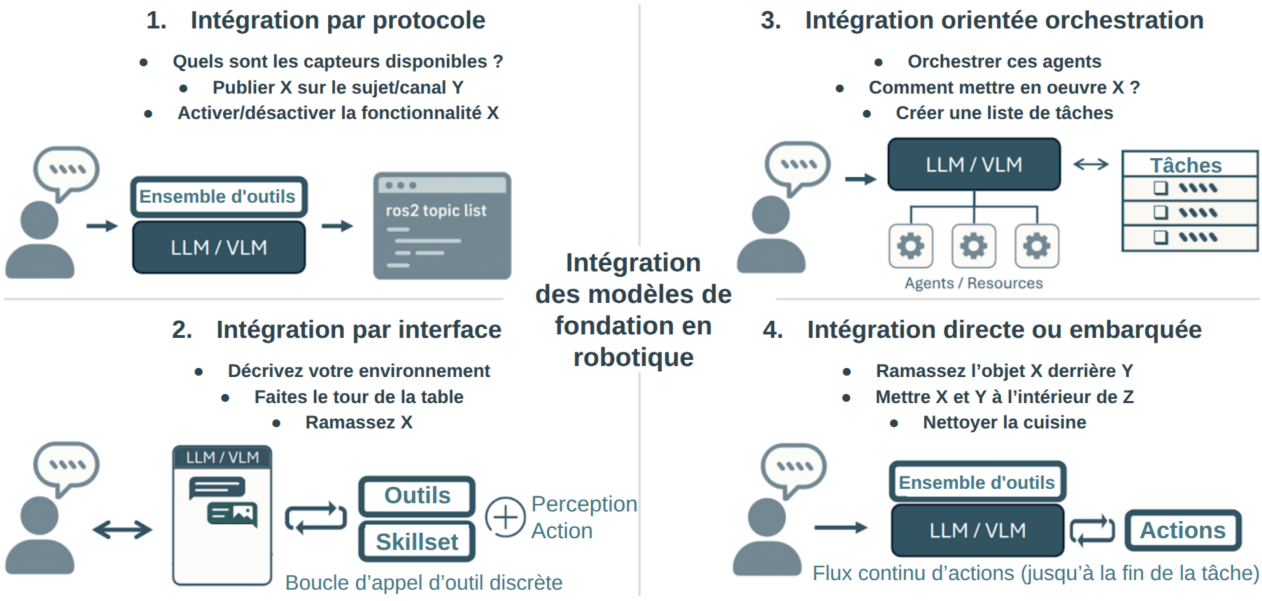


Figure 1.9 – Taxonomie des approches d'intégration des modèles de fondation en robotique. (1) L'intégration par protocole traduit les commandes utilisateur vers des appels système. (2) L'intégration par interface établit des boucles de rétroaction avec l'utilisateur. (3) L'intégration orientée orchestration coordonne plusieurs agents ou ressources. (4) L'intégration embarquée génère directement un flux continu d'actions. (Salimpour et coll. 2025)

a) Intégration par protocole

L'intégration par protocole représente l'approche conceptuellement la plus simple. Le modèle de fondation agit comme un traducteur entre les entrées utilisateur en langage naturel et un ensemble d'outils prédéfinis. L'interaction demeure unidirectionnelle : l'utilisateur soumet une commande, le modèle la traduit en appel système, et la réponse est présentée à l'utilisateur.

Des exemples représentatifs incluent `ros2ai` (Fujita 2024), qui traduit des requêtes en langage naturel vers l'interface CLI de ROS 2, et les intégrations récentes exploitant le *Model Context Protocol* (MCP) (Ye et coll. 2023 ; robotmcp 2024). L'approche *Code as Policies* (Liang et coll. 2023) s'inscrit également dans cette catégorie, où le modèle génère du code Python utilisant une API prédéfinie pour les actions sensorimotrices.

b) Intégration par interface

L'intégration par interface enrichit l'approche précédente en établissant des oucles de rétroaction. La sortie des actions influence les commandes futures ou déclenche des appels d'outils supplémentaires. L'accent est mis sur l'interaction utilisateur, notamment dans les applications d'interaction humain-robot conversationnelles (Salimpour et coll. 2025).

ROSA (*Robot Operating System Agent*) (Royce et coll. 2025) implémente un agent LLM basé sur le paradigme ReAct, abstrayant les opérations ROS en fonctions Python invocables comme outils. **RAI** (Rachwał et coll. 2025) propose une architecture distribuée où des agents spécialisés (perception, planification, contrôle, sécurité) collaborent pour une exécution concurrente. **BUMBLE** (Shah et coll. 2024) utilise un VLM unifié pour la manipulation mobile intégrant perception en monde ouvert et mémoire à double couche.

c) Intégration orientée orchestration

Cette approche se distingue par son focus sur la gestion des ressources plutôt que sur l'interaction utilisateur directe. Les modèles de fondation agissent comme planificateurs ou coordinateurs, souvent dans des contextes multi-agents ou multi-robots.

AutoRT (Ahn et coll. 2024) déploie un LLM pour orchestrer une flotte de manipulateurs mobiles, utilisant un raisonnement par prompts pour mapper des instructions vers des invocations de compétences spécifiques. **LABOR Agent**

(Chu et coll. 2024) permet la manipulation bimanuelle en sélectionnant parmi des centaines de compétences pré-entraînées selon les instructions de l'utilisateur.

d) Intégration directe ou embarquée

Cette catégorie regroupe les approches où le modèle agit comme une politique de contrôle unifiée et génère des commandes motrices de bout en bout à partir des entrées sensorielles, sans interface utilisateur explicite. Le modèle ne produit pas de réponse textuelle et n'invoque pas d'outils externes : il émet directement les signaux de commande destinés aux actionneurs. Les VLA, présentés en détail dans la Section 1.4, constituent l'architecture dominante de cette catégorie (Salimpour et coll. 2025).

1.7.4 Positionnement de notre contribution

Notre système proposé s'inscrit dans une architecture hybride combinant deux approches d'intégration complémentaires selon la taxonomie présentée :

- L'**intégration par interface** repose sur un agent LLM implémentant le paradigme ReAct, assurant l'interprétation des intentions utilisateur exprimées en langage naturel, la gestion du contexte conversationnel via un module de mémoire, et l'invocation dynamique d'outils spécialisés (reconnaissance vocale, synthèse vocale, détection de défauts).
- L'**intégration embarquée** exploite un VLA en tant que politique de contrôle, transformant directement les observations visuelles et les instructions sémantiques en commandes motrices.

Cette architecture à deux niveaux présente plusieurs avantages. L'agent LLM conserve des représentations intermédiaires explicites, facilitant l'introspection et la correction humaine en cas d'erreur de planification. Le VLA, quant à lui, assure une exécution motrice fluide et réactive, capable de s'adapter aux perturbations environnementales sans reprogrammation explicite. La séparation des préoccupations permet également une évolution indépendante des deux couches : l'agent peut être enrichi de nouveaux outils sans modifier la politique de contrôle, et inversement, le VLA peut être affiné sur de nouvelles tâches sans impacter l'interface utilisateur (Salimpour et coll. 2025 ; Yang et coll. 2025).

1.8 Conclusion

Ce chapitre a établi les fondations théoriques nécessaires à la compréhension de notre système d'interaction humain-robot. Nous avons d'abord détaillé les principes des LLM et des Transformers (Section 1.2), puis l'extension multimodale proposée par les VLM (Section 1.3). Nous avons ensuite synthétisé l'état de l'art des VLA (Section 1.4), en mettant l'accent sur leurs architectures, leurs objectifs d'entraînement (diffusion, flow matching, tokenisation d'actions) et leurs protocoles d'évaluation. Les méthodes de vision industrielle pour la détection de défauts ont été présentées (Section 1.5), suivies du paradigme d'agent (Section 1.6) permettant l'orchestration de systèmes autonomes. Enfin, nous avons analysé l'évolution des interfaces humain–robot et les différentes stratégies d'intégration des modèles de fondation (Section 1.7).

Sur cette base, le chapitre 2 présente la conception et l'architecture globale de notre système, depuis l'interface vocale jusqu'à l'orchestration par agent LLM et l'exécution des actions par un VLA, en incluant l'intégration d'outils de perception dédiés à l'inspection industrielle.

Chapitre 2

Conception et architecture du système interaction humain robot

2.1 Introduction

Dans ce chapitre, nous détaillons la conception et l'architecture globale de notre système d'interaction humain-robot. L'objectif est de présenter une vue d'ensemble de la chaîne de traitement, depuis l'émission d'une commande vocale par l'utilisateur jusqu'à l'exécution physique de la tâche par le robot et son retour verbal. Nous décrirons d'abord l'architecture globale du système, en mettant en évidence l'interaction entre les différents modules. Ensuite, nous détaillerons chaque sous-système : l'interface humain-robot, le système d'orchestration basé sur un LLM, et le système de commande du robot. Enfin, nous présentons les protocoles de communication utilisés entre les différents composants, suivis par nos choix matériels et logiciels, ainsi que l'infrastructure de calcul qui ont permis le développement et déploiement de cette architecture.

2.2 Architecture globale du système

La figure 2.1 illustre le flux de traitement complet, conçu pour permettre une interaction vocale naturelle et bidirectionnelle avec le robot.

Le pipeline de traitement commence par la capture de l'entrée vocale de l'utilisateur, qui traverse successivement les modules de détection d'activité vocale et de reconnaissance vocale pour être convertie en texte. Cette requête textuelle est ensuite transmise au système d'orchestration LLM qui, enrichi par le prompt système, la mémoire conversationnelle et la description des outils accessibles, génère une réponse appropriée et des commandes d'exécution. Les commandes sont envoyées aux outils de contrôle pour exécution, tandis qu'un mécanisme de feedback permet d'informer l'agent du résultat. Finalement, la réponse textuelle générée par l'agent est normalisée puis convertie en parole synthétique pour être restituée à l'utilisateur. Nous détaillons ci-dessous les trois sous-systèmes principaux :

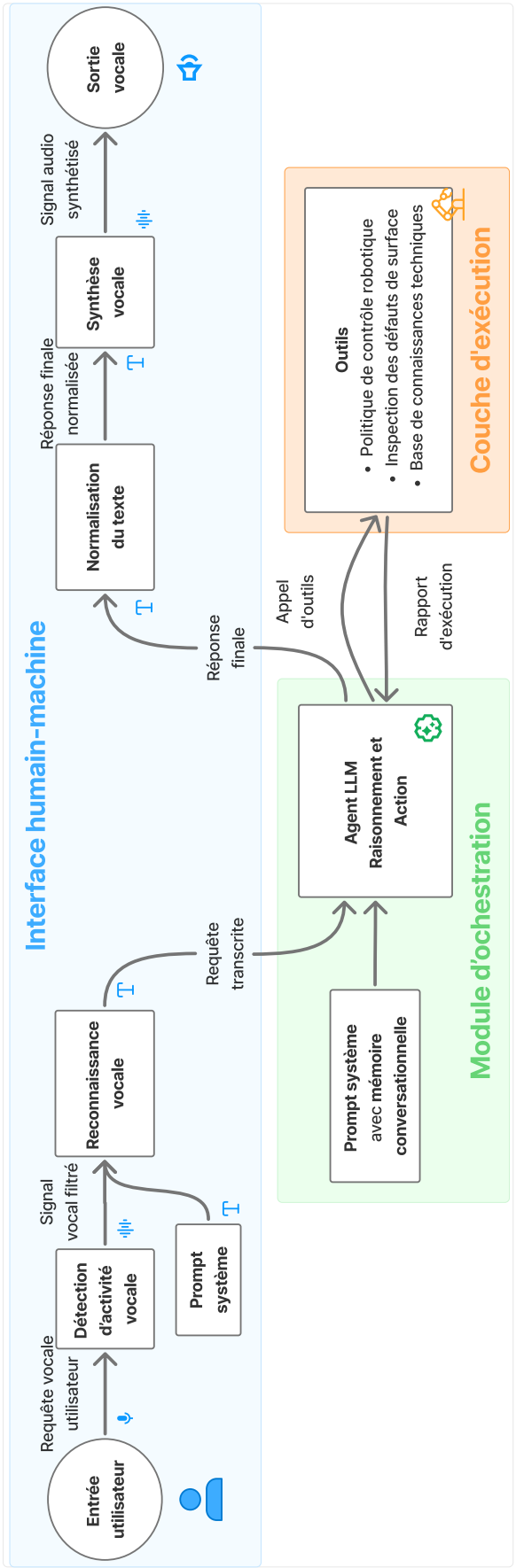


FIGURE 2.1 – Architecture du système d'orchestration LLM pour l'interaction vocale humain-robot

2.2.1 Interface humain-machine

L'interface humain-machine constitue le point de contact direct avec l'utilisateur et gère l'ensemble du cycle d'interaction vocale, de la capture de la parole jusqu'à la génération de la réponse.

a) Détection d'Activité Vocale (VAD)

Le module de détection d'activité vocale utilise SileroVAD (Team 2024), un modèle basé sur l'apprentissage profond, conçu pour identifier les segments de parole dans un flux audio en temps réel avec une faible latence et une haute précision. Il effectue une analyse continue du signal audio et détermine les moments où l'utilisateur commence et cesse de parler, filtrant efficacement les bruits de fond et les silences. Cela permet d'optimiser les ressources de calcul et d'améliorer la réactivité du système.

b) Reconnaissance vocale

Le module de reconnaissance vocale (ASR) reçoit les segments audio filtrés. Contrairement aux modèles traditionnels, ce module repose sur un modèle génératif multimodal capable de recevoir un prompt contextuel. Ce contexte aide le modèle à mieux interpréter le flux audio : face à une incertitude, il privilégie les tokens textuels les plus probables au regard du contexte fourni, garantissant une transcription plus précise et cohérente avec le domaine d'application.

c) Normalisation de texte

Ce module traite la réponse générée par l'agent avant sa conversion en parole. Il effectue des transformations pour rendre le texte adapté à la synthèse vocale (conversion des abréviations, gestion des nombres, ajustement de la ponctuation), assurant que la réponse finale soit prononcée de manière naturelle.

d) Synthèse vocale (TTS)

Le module de synthèse vocale convertit la réponse textuelle en signal audio. Il exploite un modèle génératif conçu pour la conversation, capable d'adapter dynamiquement l'intonation et le rythme pour une expressivité naturelle.

Une fois la requête de l'utilisateur capturée et encodée en texte, elle est transmise au système d'orchestration.

2.2.2 Module d'orchestration

Ce module central assure l'interprétation sémantique, la planification et la génération des réponses.

a) Prompt Système et Mémoire Conversationnelle

Le prompt système définit le contexte, les capacités et le comportement de l'agent. La mémoire conversationnelle stocke l'historique des échanges, permettant à l'agent de maintenir la cohérence contextuelle et de comprendre les références aux interactions passées durant la session.

b) Agent (LLM + Système de Raisonnement et d'Action)

Le paradigme Agent ReAct (Reasoning and Acting) étend les capacités du LLM en structurant la résolution de problèmes sous forme d'une boucle dynamique : `Pensée` → `Action` → `Observation`. L'agent analyse la requête pour formuler un plan via des étapes de raisonnement (génération de traces de pensée), sélectionne et exécute les outils nécessaires via des étapes d'action (appels de fonctions), et intègre les sorties de ces outils (observation) dans son contexte. Cette approche permet au LLM d'interagir de manière itérative avec son environnement via des outils externes mis à sa disposition pour collecter plus d'informations ou exécuter des opérations avant de générer une réponse finale.

2.2.3 Couche d'exécution

Ce module gère l'interaction du système avec son environnement, englobant le contrôle physique du bras robotique, la perception visuelle et l'accès aux ressources numériques, ainsi que la gestion des retours d'état.

Le module d'outils met à disposition de l'agent un ensemble de capacités étendues pour percevoir et agir. Ces capacités sont encapsulées dans des fonctions dédiées couvrant le contrôle robotique via des politiques VLA, l'interrogation de documentation technique et la détection de défauts en temps réel. L'agent exploite ces ressources via des appels de fonctions, qu'il peut invoquer de manière autonome et itérative en déterminant les arguments requis. Chaque fonction renvoie un rapport détaillé sur l'exécution, les modifications de l'environnement ou les erreurs rencontrées, assurant ainsi une boucle de rétroaction robuste pour la prise de décision.

2.3 Plateforme robotique SO-101

Pour la réalisation de notre preuve de concept, nous avons sélectionné le bras robotique SO-101 développé conjointement par Hugging Face et TheRobotStudio. Ce choix répond au besoin d'une plateforme d'expérimentation sécurisée et accessible, permettant de valider nos approches de contrôle robotique basées sur des modèles génératifs d'IA dont les sorties ne sont pas toujours prévisibles, sans exposer le personnel ou l'équipement aux risques associés aux robots industriels conventionnels.

Le SO-101 est un kit robotique conçu pour la téléopération et l'apprentissage par imitation dans le domaine de la robotique assistée par intelligence artificielle (Hugging Face 2025). Lancé en avril 2025 par le groupe LeRobot de Hugging Face, cette plateforme constitue une solution récente et accessible pour la recherche en robotique d'apprentissage.

2.3.1 Configuration du système

La figure 2.2 présente l'environnement complet du système incluant les deux bras robotiques ainsi que les caméras embarquées et externes.

Ce système comprend un bras guide (*leader*) et un bras suiveur (*follower*), facilitant la collecte de données pour l'entraînement de modèles d'IA : l'opérateur manipule le bras guide, tandis que le bras suiveur reproduit ses mouvements en temps réel. Lors de l'inférence, le bras suiveur est contrôlé par les politiques entraînées de manière autonome.

a) Caractéristiques générales

Les spécifications techniques du système sont les suivantes :

- **Configuration** : 2 Bras (guide + suiveur), 6 axes chacun (5 axes de mouvement + 1 pour la pince)
- **Capacité de charge** : Maximum recommandé de 400 g en bout d'effecteur (Seeed Studio 2025)
- **Espace de travail** : Portée maximale de 350 mm (Seeed Studio 2025)
- **Méthode de contrôle** : Contrôlé par ordinateur (PC, Raspberry Pi, NVIDIA Jetson)

2.3.2 Composants matériels

Le SO-101 utilise deux catégories de composants essentiels : les servomoteurs pour l'actionnement et les caméras pour la perception, détaillés ci-dessous.

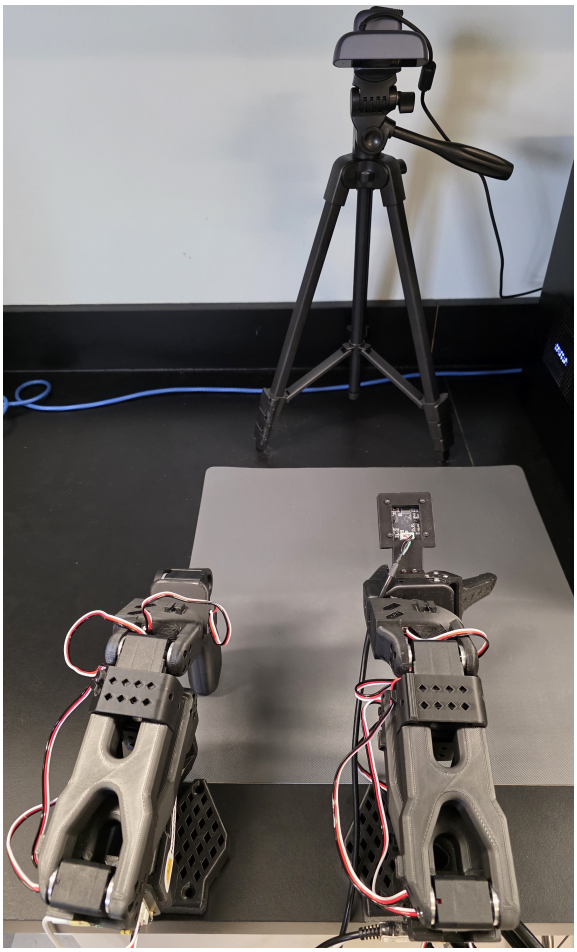


FIGURE 2.2 – Configuration complète du système SO-101 disponible dans notre laboratoire, avec les bras guide et suiveur, la caméra embarquée sur l'effecteur et la caméra externe

a) Servomoteurs

Le système utilise des servomoteurs Feetech STS3215. Le bras suiveur emploie 6 servos identiques (rapport 1 :345), tandis que le bras guide combine plusieurs rapports pour équilibrer la charge mécanique (HUGGING FACE 2025). Le tableau 2.1 détaille la répartition des moteurs et leurs ratios de réduction.

Comme le montre le tableau 2.1, la différenciation des ratios de réduction entre les deux bras répond à des contraintes fonctionnelles distinctes : le bras guide privilégie la maniabilité avec des ratios plus faibles pour faciliter la manipulation par l'opérateur, tandis que le bras suiveur adopte des ratios uniformément élevés pour maximiser la précision et la répétabilité lors de l'exécution autonome.

Les servomoteurs présentent des caractéristiques techniques essentielles au fonctionnement précis du système :

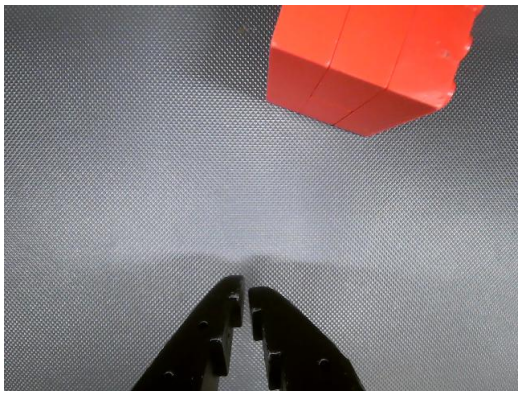
Tableau 2.1 – Spécifications des moteurs du SO-101

Composant	Bras guide	Bras suiveur
Base/Pan (J1)	1x Moteur 7.4 V, ratio 1 :191	1x Moteur 12 V, ratio 1 :345
Épaule (J2)	1x Moteur 7.4 V, ratio 1 :345	1x Moteur 12 V, ratio 1 :345
Coude (J3)	1x Moteur 7.4 V, ratio 1 :191	1x Moteur 12 V, ratio 1 :345
Flexion Poignet (J4)	1x Moteur 7.4 V, ratio 1 :147	1x Moteur 12 V, ratio 1 :345
Roulis Poignet (J5)	1x Moteur 7.4 V, ratio 1 :147	1x Moteur 12 V, ratio 1 :345
Pince (J6)	1x Moteur 7.4 V, ratio 1 :147	1x Moteur 12 V, ratio 1 :345

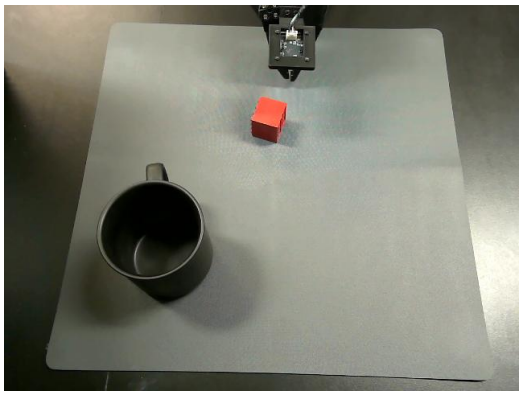
— **Résolution angulaire** : 360° (0–4096 positions/tour) via des capteurs à encodeur magnétique 12 bits (Seeed Studio 2025)
— **Alimentation** : 5 V 4 A pour le bras guide et 12 V 2 A à 5 A pour le bras suiveur (Seeed Studio 2025)
— **Couple** : Bras guide d'environ 16.5–19.5 $kg{\cdot}cm$, bras suiveur d'environ 30 $kg{\cdot}cm$ (WowRobo Robotics 2025)
— **Cartes de contrôle** : 2 cartes Seeed Studio XIAO ESP32C3 avec Bus Servo Driver Board intégrée

b) Système de vision

La perception visuelle constitue un élément crucial pour l'apprentissage et l'exécution de politiques robotiques. Notre configuration exploite deux caméras aux rôles complémentaires, offrant à la fois une vue globale de la scène et une observation détaillée de la zone de manipulation. La figure 2.3 illustre ces deux perspectives.



(a) Vue de la caméra embarquée (Wrist)



(b) Vue de la caméra externe (Front)

Figure 2.3 – Système de vision du SO-101

— **Caméra embarquée** : Comme illustré par la figure 2.3a, ce module d'environ 2 MP est situé sur l'effecteur et est dédié à la vision rapprochée et à l'alignement fin avec les objets (Seeed Studio 2025).
— **Caméra externe** : Comme le montre la figure 2.3b, cette caméra UGREEN (2K@30FPS, capteur CMOS 4 MP, FOV 80°) montée sur trépied offre une vue d'ensemble de l'espace de travail, essentielle pour la détection initiale et la planification de trajectoire.

2.3.3 Avantages du système

Au-delà de ses caractéristiques techniques, le choix du SO-101 s'appuie sur plusieurs avantages facilitant la recherche et le développement :

— **Open-source** : Solution économique principalement imprimable en 3D (fichiers STL disponibles), facilitant la reproduction et la personnalisation
— **Intégration native avec LeRobot** : Compatibilité directe avec la plateforme de Hugging Face (Hugging Face 2025)
— **Modularité** : Support de capteurs et modules supplémentaires
— **Communauté active** : Support et développement continu

2.4 Flux de données et protocoles de communication

Le système combine traitement local (pour la réactivité et le contrôle matériel) et services cloud (pour l'intelligence cognitive). Les flux de données et protocoles se répartissent comme suit :

2.4.1 Communication Interne et Matérielle

— **Inférence Locale** Les modules de détection d'activité vocale (Silero VAD), de détection d'objets (YOLO) et les VLAs pour le contrôle robotique s'exécutent localement en mémoire. L'acquisition et la restitution audio, ainsi que la capture vidéo des caméras, sont également traitées localement. Ces composants communiquent via des appels de fonctions Python directs, minimisant ainsi la latence pour ces composants.
— **Communication Série avec les Servomoteurs** La communication entre l'ordinateur et les servomoteurs du SO-101 s'appuie sur une architecture contrôleur-périphérique via un bus série partagé. Le PC transmet les commandes générées par le code Python vers une carte Bus Servo Driver, qui convertit le signal USB en UART TTL. Cette communication repose

sur le protocole Feetech, permettant l'adressage individuel ou la diffusion (*broadcast*) de commandes à l'ensemble des moteurs connectés en série. Une description technique détaillée du protocole de communication, incluant la structure des trames UART et le format de paquets, est présentée en annexe A.1, et le tableau A.1 de cette annexe fournit le tableau de contrôle complèt.

2.4.2 Communication Externe (Cloud)

Pour les interactions avec les services cloud, nous adoptons deux protocoles distincts adaptés aux exigences spécifiques de latence et de traitement de chaque composant. La figure 2.4 présente une comparaison schématique de ces deux approches architecturales.

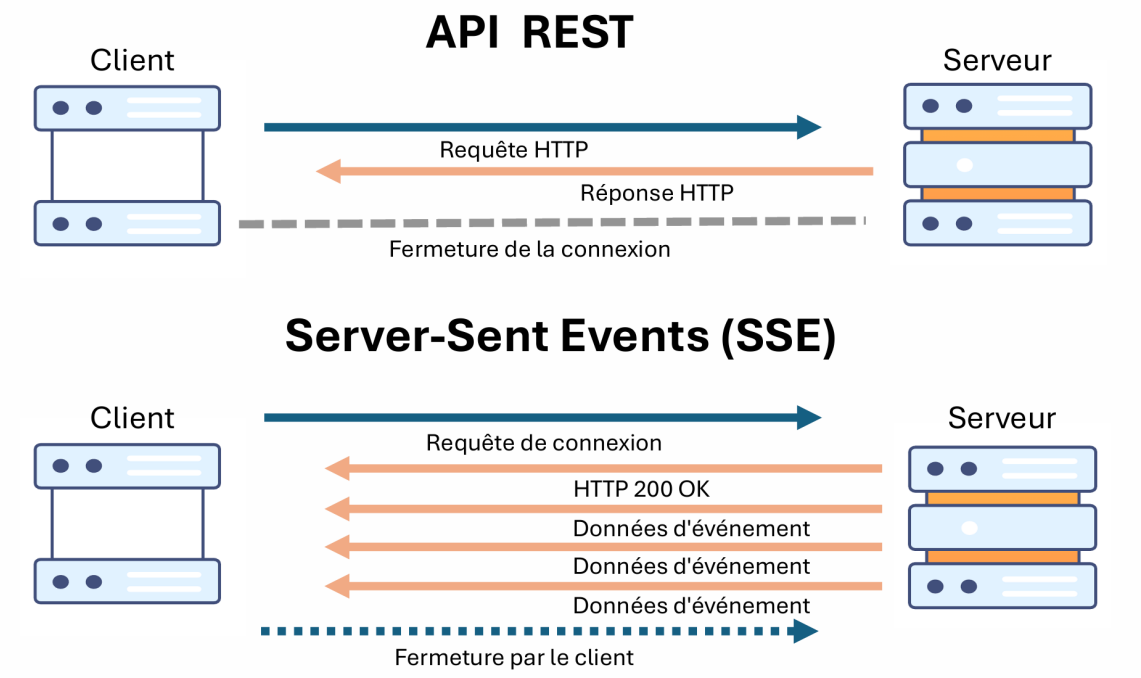


FIGURE 2.4 – Comparaison des protocoles REST et Server-Sent Events (SSE)

— **API LLM et Orchestration (SSE)** Pour les interactions avec les LLM (l'Agent principal et le modèle de normalisation), nous utilisons le protocole *Server-Sent Events* (SSE). Contrairement à une requête REST classique qui attend la complétion totale de la réponse avant de la transmettre, le SSE permet d'établir un flux unidirectionnel persistant où le serveur envoie les données (tokens) au fur et à mesure de leur génération. Cette approche de *streaming* est cruciale pour l'expérience utilisateur, car elle réduit

considérablement la latence perçue : le système peut commencer à traiter et énoncer le début d'une phrase pendant que la fin est encore en cours de génération.

— **Services Cognitifs (REST API)** Les modules de reconnaissance vocale (ASR) et de synthèse vocale (TTS), s'appuyant sur les services Google GenAI, utilisent une architecture REST standard sur HTTP/2. Dans ce cas, le traitement est effectué sur la totalité du contenu : les données audio encodées en base64 ou le texte complet sont envoyés, et le serveur renvoie le résultat final une fois le traitement terminé.

2.5 Infrastructure matérielle et logicielle du système

Cette section présente l'infrastructure technique complète du projet, organisée en deux volets complémentaires : l'infrastructure de calcul, qui détaille les ressources matérielles nécessaires à l'entraînement et au déploiement, et l'environnement logiciel, qui décrit les bibliothèques et outils utilisés pour implémenter l'architecture.

2.5.1 Infrastructure de calcul

Notre environnement de travail s'articule autour de deux composantes principales : une machine locale, utilisée pour le développement, le déploiement et l'interaction en temps réel avec le bras robotique, et une grappe de calcul haute performance, dédiée à l'entraînement des modèles de grande taille.

i. Machine locale La machine locale est utilisée pour le développement, les tests et l'inférence. Ses caractéristiques sont les suivantes :

— **CPU** : Intel Core i9-13900K (24 cœurs, 32 threads, 5.8 GHz)
— **GPU** : NVIDIA GeForce RTX 4090 (24 Go VRAM)
— **RAM** : 64 Go
— **Stockage** : 432G SSD NVMe (Système) + 3.6T HDD (Stockage modèles)
— **OS** : Ubuntu 22.04 LTS

ii. Grappe de calcul L'Alliance de recherche numérique du Canada (l'Alliance) joue un rôle central dans l'infrastructure de recherche numérique au pays. Elle coordonne et soutient les services de calcul informatique de pointe (CIP), de gestion des données de recherche et de logiciels de recherche. L'Alliance offre aux

chercheurs canadiens un accès à des ressources de calcul haute performance de classe mondiale (Alliance de recherche numérique du Canada 2025a).

Pour les besoins de ce projet, nous avons principalement exploité la grappe de calcul **Fir**. Fir est un système de calcul haute performance de pointe, actuellement classé 78e au TOP500 des supercalculateurs les plus puissants au monde. (Alliance de recherche numérique du Canada 2025b).

La grappe Fir dispose d'une architecture hétérogène comprenant des nœuds CPU et 160 nœuds GPU interconnectés par un réseau InfiniBand NDR haute performance. Pour nos travaux d'apprentissage profond, nous avons exploité les nœuds GPU dont les caractéristiques sont les suivantes :

— **Processeur** : 1 x AMD EPYC 9454 (Zen 4) @ 2.75 GHz (48 cœurs)
— **Accélérateurs** : 4 x GPU NVIDIA H100 SXM5 (80 Go de mémoire HBM3 chacun)
— **Mémoire** : 1125 Go de RAM

Le stockage est assuré par un système de fichiers parallèle haute performance de grande capacité (51 Po). Ces ressources ont été cruciales pour l'entraînement de nos modèles.

2.5.2 Environnement logiciel et bibliothèques

Le développement repose sur le langage Python (version 3.12.12) et intègre les bibliothèques suivantes :

a) Gestion de l'environnement

Nous utilisons **uv** (version 0.9.7), un gestionnaire de paquets et de projets Python extrêmement rapide qui remplace avantageusement les outils traditionnels comme `pip` et `virtualenv`. Dans notre projet, `uv` assure la création d'environnements virtuels isolés et reproductibles, ainsi que la résolution et l'installation rapides des dépendances définies dans un fichier `pyproject.toml`.

b) Orchestration et Agent

— **LangChain** (version 1.0.4) fournit une interface standardisée pour interagir avec différents fournisseurs de modèles de langage. Cette bibliothèque offre des abstractions de haut niveau pour la gestion des prompts et de la mémoire conversationnelle, l'intégration d'outils externes via des appels de fonctions,

ainsi que la construction rapide d'agents pré-configurés, tout en restant suffisamment flexible pour permettre une ingénierie contextuelle avancée.

— **LangGraph** (version 1.0.2) constitue le framework d'orchestration de bas niveau sur lequel repose LangChain. Contrairement aux abstractions de haut niveau qui masquent les détails d'implémentation, LangGraph offre un contrôle granulaire sur le flux d'exécution en modélisant les agents comme des graphes d'états (inspirés de Pregel et Apache Beam). Chaque nœud représente une fonction de calcul et les transitions entre états sont explicitement définies, permettant ainsi de combiner des flux de travail déterministes (séquences prédéfinies) et agentic (décisions contextuelles du LLM). Cette approche offre également une exécution durable (*durable execution*), permettant aux agents de persister à travers les défaillances et de reprendre leur exécution au point d'interruption.

Ces bibliothèques ont connu des évolutions récentes majeures. La version stable de LangChain v0.1.16 a été publiée le 11 avril 2024, tandis que les versions v1.0 de LangChain et LangGraph ont été officiellement lancées en Octobre 2025.

c) Contrôle Robotique

Pour l'interaction avec le bras robotique SO-101, nous utilisons la bibliothèque **LeRobot** (version 0.4.0) de Hugging Face (Cadene et coll. 2024). Cette bibliothèque, dont le développement a débuté en avril 2024, a connu plusieurs évolutions majeures au cours de cette thèse, aboutissant à la version v0.4.0 en Octobre 2025, bien qu'une version stable ne soit pas encore finalisée. **LeRobot** offre une infrastructure complète pour la robotique d'apprentissage, intégrant un format de dataset standardisé (`LeRobotDataset`) qui unifie la représentation des données multi-robots, une abstraction matérielle permettant de déployer le même code sur différentes plateformes, ainsi que des politiques d'apprentissage pré-implémentées. Dans notre système, elle assure le contrôle des servomoteurs, l'enregistrement de démonstrations par téléopération, et l'exécution de politiques d'apprentissage profond en temps réel, tout en garantissant la compatibilité avec les VLA émergents.

d) Vision par Ordinateur

Pour la tâche spécifique d'inspection industrielle, nous utilisons la bibliothèque **Ultralytics** (version 8.3.225) spécialisée dans l'entraînement et le déploiement des modèles de la famille YOLO (*You Only Look Once*). Elle permet à l'outil d'inspection

dans notre architecture d'exploiter ces modèles pour la détection d'objets en temps réel.

e) IA Générative Multimodale

Le SDK **Google GenAI** (version 1.49.0) est intégré pour exploiter les capacités multimodales des modèles d'IA générative Gemini, retenus pour leur excellent compromis entre performance, latence et coût, des critères essentiels pour une interaction fluide en temps réel. Il est utilisé pour :

- **La reconnaissance vocale et compréhension audio (Automatic Speech Recognition).** Contrairement aux approches traditionnelles de transcription (ASR), Gemini 2.5 Flash traite les fichiers audio nativement comme une modalité d'entrée, offrant une précision supérieure pour la transcription et la compréhension contextuelle.
- **La synthèse vocale (Text-to-Speech).** Le système utilise le modèle `gemini-2.5-flash-tts`, optimisé pour une génération audio à très faible latence.

f) Autres outils clés

- **Silero VAD** (version 6.2.0) : Bibliothèque utilisée pour le module de détection d'activité vocale.
- **OpenRouter** : Fournit une API unifiée offrant l'accès à des centaines de modèles d'IA via un point de terminaison unique. Agissant comme un proxy intelligent qui gère automatiquement les mécanismes de basculement (fallback) et route dynamiquement les requêtes vers les options optimales en termes de coût, latence et débit.

2.6 Conclusion

Ce chapitre a établi les fondations conceptuelles et techniques de notre système d'interaction humain-robot, en décrivant son architecture globale structurée autour de trois sous-systèmes principaux : l'interface humain-machine, le module d'orchestration LLM et le système de commande robotique.

Le choix de la plateforme SO-101 a été justifié par sa compatibilité native avec l'écosystème LeRobot et son caractère open-source permettant une expérimentation sécurisée. Nous avons ensuite détaillé les flux de données et protocoles de communication, soulignant l'importance d'une architecture hybride combinant

traitement local et services cloud pour répondre aux exigences de réactivité et de performance. Enfin, l'infrastructure matérielle et logicielle a été présentée, mettant en avant les choix technologiques clés qui la soutiennent.

Ayant établi l'architecture globale et validé la plateforme matérielle SO-101, le chapitre 3 détaillera la démarche pour transformer ce bras robotique en un agent autonome capable de manipulation physique, en entraînant et en intégrant des modèles Vision-Langage-Action (VLA).

Chapitre 3

Intégration du modèle VLA pour le contrôle robotique

3.1 Introduction

Dans ce chapitre, nous détaillons le processus d'intégration du VLA pour le contrôle du robot SO-101. Ce modèle assure la génération des actions motrices nécessaires à l'exécution autonome de tâches de manipulation à partir d'observations visuelles et d'instructions en langage naturel.

Nous couvrons l'ensemble de la chaîne d'intégration, depuis la préparation de la plateforme robotique jusqu'au déploiement et à l'évaluation des politiques apprises. Tout d'abord, nous présentons les étapes de calibration des bras robotiques guide et suiveur, étape cruciale pour garantir la précision et la reproductibilité des mouvements. Ensuite, nous décrivons le processus de collecte et de traitement des données au format LeRobotDataset, nécessaire à l'entraînement du modèle. Nous détaillons par la suite la configuration d'entraînement, incluant les stratégies d'optimisation, les fonctions de perte et les hyperparamètres utilisés.

Une étude comparative des différents modèles VLA de l'état de l'art est menée afin de sélectionner l'architecture la plus adaptée au déploiement de notre système de contrôle robotique autonome. Nous présentons le système d'évaluation standardisé mis en place pour tester les politiques apprises dans des conditions contrôlées, permettant de mesurer objectivement leur taux de réussite et leur robustesse. Enfin, nous discutons des limitations observées concernant la généralisation et l'intégration des modèles VLA probabilistes dans un contexte robotique réel, identifiant les défis techniques et les contraintes matérielles rencontrés.

3.2 Calibration du robot SO-101

La précision et la reproductibilité des mouvements reposent sur une modélisation fidèle de la cinématique du robot et une calibration rigoureuse de ses actionneurs. Cette section décrit le processus de transformation bidirectionnelle entre les

positions physiques des moteurs et les valeurs normalisées utilisées par les modèles d'apprentissage profond. Nous détaillons ensuite le système d'encodage angulaire, la description cinématique complète du bras, les mécanismes de calibration et de limitation articulaire, ainsi que les procédures de normalisation permettant l'interfaçage avec les architectures VLA.

3.2.1 Encodage et résolution angulaire

Les moteurs Feetech STS3215 équipant le bras SO-101 disposent d'un encodeur magnétique absolu sur 12 bits. Cet encodeur convertit la position angulaire de l'axe moteur en une valeur numérique entière entre 0 et 4095, correspondant aux 2^{12} positions discrètes possibles sur une rotation complète.

La relation entre l'angle physique θ (en degrés) et la valeur brute de l'encodeur P_{raw} s'exprime par la correspondance suivante :

$$360^\circ \longleftrightarrow 4096 \text{ pas} \tag{3.1}$$

La résolution angulaire du système est donc :

$$\Delta\theta = \frac{360^\circ}{4096} \approx 0.088^\circ \text{ par pas} \tag{3.2}$$

Cette résolution permet d'atteindre une précision de positionnement inférieure au dixième de degré, suffisante pour les tâches de manipulation fine envisagées. La conversion entre valeur brute et angle physique se calcule ainsi :

$$\theta = P_{\text{raw}} \times \frac{360}{4096} \approx P_{\text{raw}} \times 0.088^\circ \tag{3.3}$$

3.2.2 Description cinématique et articulations

Le bras robotique SO-101 possède 6 degrés de liberté (DoF), chacun contrôlé par un moteur identifié par un ID unique. La chaîne cinématique est structurée comme suit :

— **Rotation en lacet de l'épaule (*Shoulder Pan*) (ID 1)** : Rotation de la base autour de l'axe vertical. Ce mouvement permet l'orientation gauche-droite du bras dans le plan horizontal, définissant l'espace de travail azimutal du robot.

— **Élévation de l'épaule (*Shoulder Lift*) (ID 2)** : Élévation du bras supérieur. Cette articulation contrôle le mouvement vertical de l'ensemble du bras, déterminant la hauteur d'opération de l'effecteur terminal.
— **Flexion du coude (*Elbow Flex*) (ID 3)** : Flexion du coude. Ce joint permet de replier l'avant-bras vers le bras supérieur, réduisant ou augmentant l'extension totale du bras pour atteindre différentes profondeurs dans l'espace de travail.
— **Flexion du poignet (*Wrist Flex*) (ID 4)** : Inclinaison du poignet dans le plan vertical. Cette articulation oriente l'effecteur terminal vers le haut ou vers le bas, permettant d'approcher les objets selon différents angles d'attaque.
— **Rotation en roulis du poignet (*Wrist Roll*) (ID 5)** : Rotation du poignet autour de son axe longitudinal. Ce mouvement permet la rotation de la pince sans modifier sa position spatiale, essentiel pour les tâches nécessitant une orientation précise de l'objet saisi.
— **Pince (*Gripper*) (ID 6)** : Ouverture et fermeture de la pince. Cet actionneur contrôle la préhension des objets. La plage de mouvement de cette articulation détermine la taille maximale des objets manipulables.

3.2.3 Calibration et offset de référence

La procédure de calibration aligne le modèle numérique du robot avec sa configuration physique. Chaque articulation est d'abord positionnée au milieu de sa plage de mouvement, puis déplacée manuellement jusqu'à ses limites physiques dans les deux directions (positive et négative), déterminant ainsi les bornes P_{min} et P_{max}.

Pour compenser les variations mécaniques d'assemblage (alignement des cornes de servomoteurs, jeu dans les liaisons, tolérances de montage), un décalage de référence (*Homing Offset*) P_{offset} est enregistré pour chaque moteur. Ce paramètre, correspondant à la lecture brute de l'encodeur P_{raw} dans la configuration médiane, définit le zéro virtuel de l'articulation. Les trois paramètres de calibration (P_{offset}, P_{min}, P_{max}) sont stockés dans la mémoire non volatile (EEPROM) du moteur et servent de référence pour toutes les opérations ultérieures.

La position calibrée P_{present} se calcule par simple soustraction de l'offset :

$$P_{\text{present}} = P_{\text{raw}} - P_{\text{offset}} \tag{3.4}$$

Cette transformation établit un référentiel cohérent où la position zéro correspond à la configuration géométrique de référence, indépendamment des

variations d'assemblage. Les bornes calibrées préviennent les collisions mécaniques en contraignant toute position lue dans l'intervalle de sécurité :

$$P_{\text{clamped}} = \min(P_{\max}, \max(P_{\min}, P_{\text{present}})) \tag{3.5}$$

Le tableau 3.1 et le tableau 3.2 présentent les paramètres de calibration pour les deux bras du système SO-101.

TABLEAU 3.1 – Paramètres de calibration du bras suiveur

Articulation	**ID**	**Décalage**	**Min**	**Max**	**Mode**
Rotation en lacet de l'épaule	1	-207	709	3429	0
Élévation de l'épaule	2	-662	799	3159	0
Flexion du coude	3	1838	960	3167	0
Flexion du poignet	4	1083	800	3105	0
Rotation en roulis du poignet	5	-168	0	4095	0
Pince	6	-33	2030	3543	0

TABLEAU 3.2 – Paramètres de calibration du bras de téléopération (Teleop)

Articulation	**ID**	**Décalage**	**Min**	**Max**	**Mode**
Rotation en lacet de l'épaule	1	1747	665	3350	0
Élévation de l'épaule	2	1856	870	3239	0
Flexion du coude	3	44	883	3092	0
Flexion du poignet	4	482	880	3193	0
Rotation en roulis du poignet	5	549	224	4061	0
Pince	6	-2014	2032	3281	0

Le paramètre sens de rotation (*Drive Mode*) indique l'orientation du moteur : une valeur de 0 correspond à l'orientation standard, tandis qu'une valeur de 1 inverse le sens de rotation pour s'adapter aux contraintes mécaniques d'assemblage.

3.2.4 Normalisation des actions

Pour l'entraînement des modèles VLA et l'inférence, les positions articulaires brutes sont normalisées dans un intervalle standardisé. Cette normalisation facilite l'apprentissage en uniformisant les échelles de valeurs et permet la généralisation du modèle à travers différentes configurations physiques et calibrations.

a) Articulations du bras (Shoulder, Elbow, Wrist)

Ces articulations sont configurées en mode `RANGE_M100_100`, mappant linéairement l'intervalle $[P_{\min}, P_{\max}]$ vers l'intervalle $[-100, 100]$. L'équation de normalisation s'écrit :

$$V_{\text{norm}}^{\text{joint}} = \left(\frac{P_{\text{clamped}} - P_{\min}}{P_{\max} - P_{\min}} \right) \times 200 - 100 \tag{3.6}$$

Si le paramètre sens de rotation est défini sur 1 (orientation inversée), le signe du résultat est inversé :

$$V_{\text{final}}^{\text{joint}} = \begin{cases} V_{\text{norm}}^{\text{joint}} & \text{si mode} = 0 \\ -V_{\text{norm}}^{\text{joint}} & \text{si mode} = 1 \end{cases} \tag{3.7}$$

b) Pince

Le moteur de la pince est configuré en mode `RANGE_0_100`, mappant l'intervalle $[P_{\min}, P_{\max}]$ vers l'intervalle $[0, 100]$. L'équation de normalisation devient :

$$V_{\text{norm}}^{\text{gripper}} = \left(\frac{P_{\text{clamped}} - P_{\min}}{P_{\max} - P_{\min}} \right) \times 100 \tag{3.8}$$

Cette normalisation offre deux avantages pour l'intégration avec les modèles VLA :

— Abstrait les détails matériels spécifiques au bras robotique, permettant au modèle d'apprendre des représentations de mouvement indépendantes des valeurs d'encodeur brutes.

— Permet la portabilité des politiques apprises entre différents exemplaires du robot SO-101, même avec des paramètres de calibration différents.

3.2.5 Pipeline de transformation bidirectionnelle

Le pipeline de transformation décrit le flux complet des données entre le robot physique et le modèle VLA, dans les deux sens : collecte de démonstrations utilisée pour l'entraînement des modèles (robot vers dataset) et déploiement de politiques (modèle vers robot).

a) Collecte de données : du robot vers l'ensemble de données

Lors de l'enregistrement de démonstrations par téléopération, les positions physiques sont converties en valeurs normalisées selon la séquence suivante. Illustrons ce processus sur l'articulation rotation en lacet de l'épaule (ID 1) du bras suiveur, dont les paramètres sont : Offset = -207, Min = 709, Max = 3429 (plage de mouvement calibrée de $3429 - 709 = 2720$ pas).

1. **Lecture capteur (P_{raw})** : On interroge le moteur via le bus série pour obtenir la position actuelle de l'encodeur, valeur entière dans l'intervalle $[0, 4095]$. Supposons que le moteur se trouve à la position brute $P_{\text{raw}} = 2000$.
2. **Correction de l'offset (P_{present})** : On soustrait l'offset de calibration pour obtenir la position dans le référentiel aligné :

$$P_{\text{present}} = P_{\text{raw}} - P_{\text{offset}} \tag{3.9}$$

   Dans notre exemple : $P_{\text{present}} = 2000 - (-207) = 2207$.
3. **Limitation (clamping)** : On contraint la valeur dans les bornes de sécurité définies lors de la calibration :

$$P_{\text{clamped}} = \min(P_{\max}, \max(P_{\min}, P_{\text{present}})) \tag{3.10}$$

   Ici, $P_{\text{clamped}} = 2207$ (déjà dans les bornes $[709, 3429]$).
4. **Normalisation ($V_{\text{norm}}^{\text{joint}}$)** : La valeur est convertie en format standardisé :

$$V_{\text{norm}}^{\text{joint}} = \left(\frac{P_{\text{clamped}} - P_{\min}}{P_{\max} - P_{\min}} \right) \times 200 - 100 \tag{3.11}$$

   Application numérique :

$$\begin{aligned} V_{\text{norm}}^{\text{joint}} &= \left(\frac{2207 - 709}{2720} \right) \times 200 - 100 \\ &= \left(\frac{1498}{2720} \right) \times 200 - 100 \\ &\approx 0.5507 \times 200 - 100 = 10.15 \end{aligned} \tag{3.12}$$

Cette valeur $V_{\text{norm}}^{\text{joint}} = 10.15$ est sauvegardée dans l'ensemble de données au format Parquet, accompagnée des observations visuelles synchronisées et des métadonnées temporelles.

b) Inférence : du modèle VLA vers le robot

Le modèle VLA prédit des actions normalisées qui doivent être converties en commandes moteurs physiques selon le processus inverse. Reprenons l'exemple de l'articulation rotation en lacet de l'épaule (ID 1) du bras suiveur (Offset = -207, Min = 709, Max = 3429).

1. **Sortie du modèle (V_{norm})** : Le réseau de neurones génère une prédiction dans l'intervalle $[-100, 100]$ pour les articulations du bras ($V_{\text{norm}}^{\text{joint}}$) et $[0, 100]$ pour la pince ($V_{\text{norm}}^{\text{gripper}}$). Supposons que le modèle VLA prédit l'action $V_{\text{norm}}^{\text{joint}} = 10.15$.

2. **Dénormalisation ($P_{\text{calibrated}}$)** : On convertit la valeur normalisée en position calibrée (pas d'encodeur). Pour les articulations du bras :

$$P_{\text{calibrated}}^{\text{joint}} = P_{\min} + \frac{(V_{\text{norm}}^{\text{joint}} + 100) \times (P_{\max} - P_{\min})}{200} \tag{3.13}$$

   Application numérique :

$$\begin{aligned} P_{\text{calibrated}}^{\text{joint}} &= 709 + \frac{(10.15 + 100) \times 2720}{200} \\ &= 709 + \frac{110.15 \times 2720}{200} \\ &= 709 + 1498.04 \approx 2207 \end{aligned} \tag{3.14}$$

   Pour la pince, la formule s'adapte à l'intervalle $[0, 100]$:

$$P_{\text{calibrated}}^{\text{gripper}} = P_{\min} + \frac{V_{\text{norm}}^{\text{gripper}} \times (P_{\max} - P_{\min})}{100} \tag{3.15}$$

3. **Application de l'offset (P_{raw})** : On ajoute l'offset de calibration pour obtenir la position brute que le moteur comprend :

$$P_{\text{raw}} = P_{\text{calibrated}} + P_{\text{offset}} \tag{3.16}$$

   Dans notre exemple : $P_{\text{raw}} = 2207 + (-207) = 2000$.

4. **Commande moteur** : La valeur $P_{\text{raw}} = 2000$ (entier dans $[0, 4095]$) est transmise au contrôleur du moteur qui se déplace physiquement à l'angle correspondant :

$$\theta_{\text{physique}} \approx 2000 \times 0.088^\circ \approx 176^\circ \tag{3.17}$$

3.3 Collecte de données avec LeRobotDataset

Le format LeRobotDataset est le standard d'organisation et de stockage des données de démonstration robotique développé par Hugging Face. Cette section présente d'abord le processus de collecte par téléopération selon un protocole d'acquisition synchronisée, puis détaille l'architecture modulaire de ce format, incluant la structure hiérarchique du dataset et le contenu des différents dossiers (données tabulaires, métadonnées, vidéos).

3.3.1 Processus de collecte par téléopération

Le système de collecte exploite l'architecture guide-suiveur du SO-101, où l'opérateur humain manipule le bras guide pendant que le bras suiveur reproduit ses mouvements en temps réel. L'acquisition des données s'effectue à une fréquence fixe de 30 Hz, synchronisant l'enregistrement des positions articulaires des deux bras avec la capture des images des deux caméras.

À chaque instant d'échantillonnage, le système enregistre :

— Les positions normalisées des six articulations du bras guide (stockées dans **action**)

— Les positions normalisées des six articulations du bras suiveur (stockées dans **observation.state**)

— Une frame RGB de la caméra frontale (480×640)

— Une frame RGB de la caméra poignet (480×640)

— Le timestamp relatif depuis le début de l'épisode

— L'identifiant de la tâche en cours (`task_index`)

L'opérateur démarre l'enregistrement au début de l'épisode, exécute la tâche de manipulation via le bras guide, puis arrête l'enregistrement une fois la tâche accomplie. Le système calcule automatiquement les statistiques de l'épisode (min, max, mean, std pour chaque feature) et les sauvegarde dans les métadonnées.

3.3.2 Architecture du format LeRobotDataset v3.0

Le format LeRobotDataset v3.0 adopte une architecture modulaire conçue pour gérer efficacement des volumes massifs de données multi-modales. Contrairement aux versions précédentes où chaque épisode correspondait à un fichier distinct, cette version introduit un mécanisme de segmentation (chunking) permettant de concaténer les données de plusieurs épisodes dans des fichiers de plus grande taille,

optimisant ainsi les opérations d'entrée/sortie et facilitant la scalabilité vers des millions d'épisodes.

La structure repose sur trois dossiers principaux, chacun ayant un rôle spécifique dans l'organisation des données, comme illustré dans la figure 3.1.

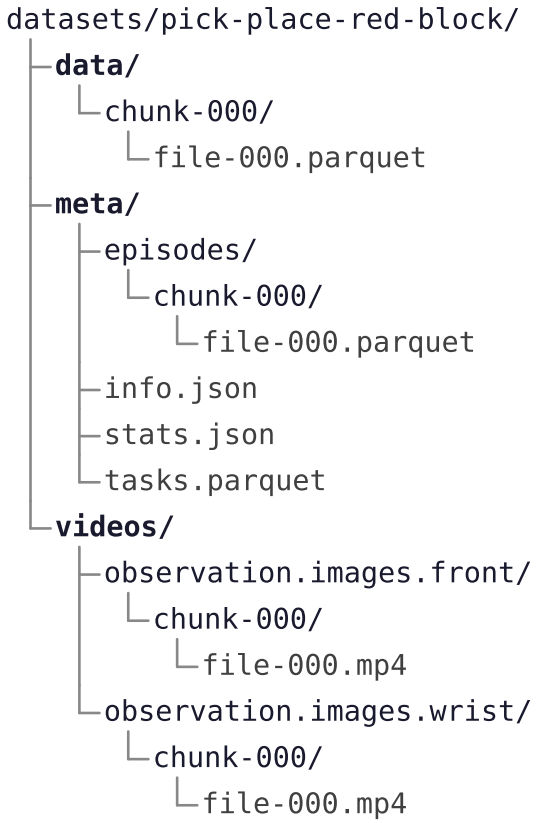


FIGURE 3.1 – Structure hiérarchique du format LeRobotDataset v3.0

Les données de multiples épisodes sont concaténées dans les mêmes fichiers Parquet et vidéo, tandis que les métadonnées d'épisodes stockent les indices de début et de fin permettant de reconstruire chaque épisode individuellement lors du chargement.

3.3.3 Dossier `data/` : séries temporelles synchronisées

Le dossier `data/` contient les fichiers Parquet stockant les séries temporelles des valeurs **observation.state** et **action**. Les données sont organisées selon un modèle frame-based, où chaque ligne du fichier représente une frame individuelle capturée à un instant d'échantillonnage donné (30 Hz dans notre cas). Le tableau 3.3 présente la

structure détaillée des colonnes et fournit des exemples de valeurs réelles extraites du troisième épisode (Episode Index 2, Frame 0) du dataset `pick-place-red-block`.

TABLEAU 3.3 – Structure des colonnes du fichier Parquet de données

Colonne	**Type**	**Description**	**Exemple (Ep. 2, Frame 0)**
action	`list<float>`	Vecteur des positions normalisées des six articulations du bras guide (cible d'apprentissage)	`[-5.10, -99.49, 93.39, 69.39, -6.02, 0.80]`
observation.state	`list<float>`	Vecteur des positions normalisées des six articulations du bras suiveur (état observé)	`[-5.44, -98.98, 93.66, 70.59, -5.84, 0.86]`
`timestamp`	`float32`	Horodatage de la frame dans l'épisode (en secondes, remis à 0 par épisode)	`0.0`
`frame_index`	`int64`	Index de la frame au sein de l'épisode (commence à 0)	`0`
`episode_index`	`int64`	Identifiant unique de l'épisode auquel appartient cette frame	`2`
`index`	`int64`	Index global de la frame dans l'ensemble du dataset concaténé	`291`
`task_index`	`int64`	Identifiant de la tâche associée (référence vers `tasks.parquet`)	`0`

La différence entre les vecteurs **action** et **observation.state** (de l'ordre de 0.3° à 1.2° par articulation) résulte de la latence du système et de l'inertie mécanique du bras suiveur, qui ne peut s'aligner instantanément sur la consigne du guide à la fréquence de contrôle de 30 Hz.

Les images capturées par les caméras sont synchronisées avec les données tabulaires via les horodatages. Le `timestamp` de chaque frame (temps relatif depuis le début de l'épisode) est additionné au `from_timestamp` stocké dans les métadonnées pour localiser précisément l'image correspondante dans le fichier vidéo concaténé.

3.3.4 Dossier `meta/` : métadonnées et statistiques

Le dossier `meta/` regroupe l'ensemble des métadonnées nécessaires à l'interprétation et à l'utilisation du dataset. Ces fichiers décrivent la configuration globale, les statistiques de normalisation, les définitions de tâches, et les caractéristiques détaillées de chaque épisode.

a) Fichier `info.json`

Ce fichier JSON contient les informations générales sur l'ensemble de données :

- `codebase_version` : Version du format LeRobotDataset utilisé pour la collecte
- `robot_type` : Type de robot (ex : `"so101"`)
- `total_episodes` : Nombre total d'épisodes dans l'ensemble de données
- `total_frames` : Nombre total de frames enregistrées
- `fps` : Fréquence d'échantillonnage
- `features` : Liste des caractéristiques enregistrées, comprenant **action** et **observation** (qui inclut `state`, `images.front`, et `images.wrist`)
- `splits` : Découpage train/validation/test

b) Fichier `stats.json`

Ce fichier stocke les statistiques calculées sur l'ensemble du dataset pour chaque feature. Pour les vecteurs **action** et **observation.state**, on trouve la moyenne (`mean`), l'écart-type (`std`), le minimum (`min`) et le maximum (`max`) pour chaque dimension. Ces statistiques sont utilisées lors du prétraitement pour normaliser les données avant l'entraînement du modèle VLA, assurant une stabilité numérique optimale.

c) Fichier `tasks.parquet`

Ce fichier liste les tâches définies dans l'ensemble de données. Chaque ligne associe un identifiant numérique `task_index` à une description textuelle en langage naturel. Par exemple, pour `pick-place-red-block`, la tâche d'index 0 correspond à la description : *"Put the red block in the cup"*.

d) Dossier `episodes/`

Ce sous-dossier contient des fichiers Parquet décrivant chaque épisode individuellement. Le tableau 3.4 présente les principales colonnes et leurs valeurs pour le troisième épisode (Episode Index 2) du dataset `pick-place-red-block`.

Les colonnes `dataset_from_index` et `dataset_to_index` permettent d'extraire les lignes correspondant à cet épisode du fichier Parquet concaténé.

TABLEAU 3.4 – Métadonnées d'épisode (exemple : Episode Index 2)

Colonne	Description	Exemple
`episode_index`	Identifiant unique de l'épisode	`2`
`tasks`	Liste des tâches accomplies dans cet épisode	`['Put the red block in the cup']`
`length`	Nombre de frames dans l'épisode	`111`
`data/chunk_index`	Index du chunk de données contenant cet épisode	`0`
`data/file_index`	Index du fichier de données	`0`
`dataset_from_index`	Frame index de début dans l'ensemble de données global	`291`
`dataset_to_index`	Frame index de fin dans l'ensemble de données global	`402`
`videos/.../from_timestamp`	Timestamp de début dans la vidéo concaténée	`9.7` (s)
`videos/.../to_timestamp`	Timestamp de fin dans la vidéo concaténée	`13.4` (s)

3.3.5 Dossier `videos/` : enregistrements visuels

Le dossier `videos/` stocke les enregistrements vidéo capturés par les caméras du robot, organisés par source et par chunk. Pour notre système, deux flux vidéo sont enregistrés simultanément :

— `observation.images.front/` : Vidéos de la caméra externe montée sur support (résolution 480×640, 30 FPS).

— `observation.images.wrist/` : Vidéos de la caméra embarquée sur l'effecteur (résolution 480×640, 30 FPS).

3.4 Entraînement du modèle VLA

L'entraînement des politiques VLA repose sur l'apprentissage supervisé à partir des démonstrations collectées. Cette section détaille l'ensemble du pipeline d'entraînement, depuis le prétraitement des observations multimodales jusqu'à la génération d'actions exécutables sur le robot. Nous présentons successivement les transformations appliquées aux données visuelles et articulaires, la configuration des hyperparamètres (optimiseur, ordonnanceur, fonction de perte), le mécanisme de génération d'actions par *Flow Matching*, et les métriques utilisées pour évaluer les performances des politiques apprises.

3.4.1 Préparation et prétraitement des données

L'étape initiale consiste à convertir les observations multimodales (images, états articulaires, instructions textuelles) en représentations numériques normalisées et compatibles avec l'architecture du modèle d'apprentissage profond.

a) Normalisation des données

La normalisation des données est une étape essentielle pour garantir la stabilité de l'apprentissage. Dans notre pipeline, les observations visuelles et les états articulaires sont normalisés en utilisant les statistiques globales calculées sur l'ensemble de données et stockées dans le fichier `stats.json`.

Les images capturées par les deux caméras sont d'abord redimensionnées à la résolution d'entrée spécifique à chaque architecture (224×224 pour SmolVLA et Pi0.5, 256×256 pour GR00T) par interpolation bilinéaire. Les valeurs de pixels sont ensuite normalisées pour chaque canal de couleur RGB :

$$I_{\text{norm}} = \frac{I_{\text{raw}}/255.0 - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \tag{3.18}$$

où $\boldsymbol{\mu}$ et $\boldsymbol{\sigma}$ sont les statistiques moyennes et écarts-types extraites du fichier de statistiques du dataset.

De même, les vecteurs **observation.state** sont normalisés par z-score :

$$s_{\text{norm}} = \frac{s_{\text{raw}} - \boldsymbol{\mu}_s}{\boldsymbol{\sigma}_s} \tag{3.19}$$

où $\boldsymbol{\mu}_s$ et $\boldsymbol{\sigma}_s$ sont respectivement la moyenne et l'écart-type calculés sur l'ensemble des frames du dataset pour chaque dimension articulaire. Cette normalisation centre les données autour de zéro avec une variance unitaire, facilitant la convergence des algorithmes d'optimisation.

b) Encodage des instructions textuelles

Les descriptions de tâches en langage naturel (ex : *"Put the red block in the black cup"*) sont tokenisées via le tokenizer du modèle de langage pré-entraîné intégré au VLA. Le texte est converti en une séquence d'identifiants de tokens, puis complété (padding) ou tronqué pour atteindre une longueur fixe : 77 tokens pour SmolVLA, 512 tokens pour pi0.5 et GR00T.

Ces tokens sont ensuite projetés dans l'espace latent du modèle de langage via une couche d'embedding, générant une représentation vectorielle continue. Cette

représentation capture la sémantique de la tâche permettant au modèle de produire des comportements distincts pour des instructions différentes.

3.4.2 Configuration de l'entraînement et hyperparamètres

Afin de comparer équitablement les architectures GR00T, SmolVLA et Pi0.5, nous avons standardisé les hyperparamètres principaux tout en conservant les valeurs par défaut pour les paramètres spécifiques à chaque architecture :

— **Taille du lot** (*batch size*) : 64 échantillons par itération
— **Nombre de pas** : 20 000 itérations d'optimisation
— **Horizon de prédiction** (*chunk size*) : 16 prédictions par itération
— **Matériel** : NVIDIA H100 80GB HBM3 *(High Memory Bandwidth)*

a) Segmentation d'actions et fenêtrage temporel

Les modèles VLA adoptent une stratégie de prédiction par blocs d'actions (*action chunking*) où une seule observation multimodale génère une séquence de C actions futures. Ce paramètre C, appelé horizon de prédiction (*chunk size*), constitue un hyperparamètre fondamental influençant à la fois les performances et la latence du système.

i. Configuration des décalages temporels Le mécanisme de fenêtrage temporel repose sur les décalages temporels relatifs (*delta timestamps*), vecteurs définissant les pas de temps pour l'échantillonnage des observations et actions. Pour un modèle configuré avec `chunk_size` $= C$ et une fréquence d'échantillonnage $f = 30$ Hz, les décalages temporels sont calculés automatiquement :

$$\Delta t_{\text{obs}} = [0.0] \quad \text{(observation courante uniquement)} \tag{3.20}$$

$$\Delta t_{\text{action}} = \left[0, \frac{1}{f}, \frac{2}{f}, \ldots, \frac{C-1}{f}\right] = \left[0.0, 0.033, 0.067, \ldots, \frac{C-1}{30}\right] \tag{3.21}$$

Cette configuration signifie qu'à chaque pas d'entraînement, le modèle reçoit :
— Une observation instantanée : $\mathbf{s}_t \in \mathbf{R}^6$ (état articulaire) et $\mathbf{I}_t \in \mathbf{R}^{3\times H\times W}$ (images)
— Une séquence cible d'actions : $\{\mathbf{a}_t, \mathbf{a}_{t+1}, \ldots, \mathbf{a}_{t+C-1}\}$ où $\mathbf{a}_{t+k} \in \mathbf{R}^6$

ii. Choix de l'horizon de prédiction Pour nos expériences, nous avons fixé $C = 16$ actions, correspondant à un horizon temporel de $16/30 \approx 0.53$ secondes. Ce choix résulte d'un compromis entre Réactivité et cohérence : Un horizon de 16 actions implique une nouvelle observation toutes les 0.53 secondes. Des horizons plus courts (ex : $C = 10$, ré-observation toutes les 0.33s) augmentent la réactivité aux variations dans l'environnement mais génèrent des trajectoires moins fluides car le modèle recalcule fréquemment sans exploiter pleinement sa capacité de planification temporelle.

iii. Processus d'entraînement Durant l'entraînement, pour un échantillon de dataset correspondant à la frame t, le chargeur de données extrait automatiquement :

$$\text{Batch} = \{(\mathbf{s}_t, \mathbf{I}_t), [\mathbf{a}_t, \mathbf{a}_{t+1}, \ldots, \mathbf{a}_{t+15}]\} \tag{3.22}$$

Le modèle apprend la politique π_θ définie par :

$$\pi_\theta : (\mathbf{s}_t, \mathbf{I}_t, \text{task}) \longrightarrow [\hat{\mathbf{a}}_t, \hat{\mathbf{a}}_{t+1}, \ldots, \hat{\mathbf{a}}_{t+15}] \tag{3.23}$$

iv. Déploiement en inférence Lors de l'exécution sur le robot, le modèle génère une séquence de 16 actions à partir d'une seule observation. Ces actions sont ensuite exécutées séquentiellement à la fréquence de contrôle de 30 Hz : l'action $\hat{\mathbf{a}}_t$ est envoyée au robot à l'instant $t = 0$, l'action $\hat{\mathbf{a}}_{t+1}$ à $t = 0.033$s, et ainsi de suite jusqu'à $\hat{\mathbf{a}}_{t+15}$ à $t = 0.5$s. Une fois toutes les actions exécutées (après 0.53 secondes), une nouvelle observation est capturée et le cycle recommence. Ce mécanisme réduit la fréquence des appels au modèle de 30 Hz à environ 1.88 Hz, diminuant significativement la charge computationnelle tout en maintenant une fluidité d'exécution grâce à la cohérence temporelle des actions prédites.

b) Fonction de perte

L'erreur quadratique moyenne (Mean Squared Error, MSE) constitue la fonction de perte principale pour l'ensemble des architectures VLA étudiées. Pour un batch de B échantillons, une séquence de $C = 16$ actions par prédiction, et des actions de dimension $D = 6$ (six articulations du robot SO-101), la perte MSE s'exprime :

$$\mathcal{L}_{\text{MSE}} = \frac{1}{B \cdot C \cdot D} \sum_{i=1}^{B} \sum_{k=0}^{C-1} \sum_{j=1}^{D} \left(\hat{a}^{(i)}_{t+k,j} - a^{(i)}_{t+k,j} \right)^2 \tag{3.24}$$

où $\hat{a}^{(i)}_{t+k,j}$ est la j-ème composante de l'action prédite à l'instant $t+k$ pour l'échantillon i du batch, et $a^{(i)}_{t+k,j}$ est la valeur de référence (ground truth) correspondante issue des démonstrations. La perte est moyennée sur l'ensemble du batch, sur toutes les actions de la séquence prédite, et sur toutes les dimensions articulaires.

c) Optimiseur AdamW

L'optimiseur AdamW (*Adam with Weight Decay*) est utilisé pour tous les entraînements. Contrairement à Adam classique, AdamW découple la régularisation L2 (*weight decay*) de l'adaptation du taux d'apprentissage, améliorant la généralisation. Les équations de mise à jour sont :

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{3.25}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{3.26}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{3.27}$$

$$w_{t+1} = w_t - \eta \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda w_t \right) \tag{3.28}$$

où g_t est le gradient, m_t et v_t sont les premiers et seconds moments, η est le taux d'apprentissage, λ est le coefficient de régularisation (*weight decay*), w_t représente les poids du modèle à l'itération t, et β_1, β_2, ϵ sont des hyperparamètres.

Les valeurs utilisées sont :

- $\eta = 2 \times 10^{-5}$ (taux d'apprentissage initial)
- $\lambda = 0.0$ (régularisation désactivée, conformément aux configurations par défaut des VLA sélectionnés pour l'ajustement fin)
- $\beta_1 = 0.9$, $\beta_2 = 0.95$
- $\epsilon = 10^{-8}$

d) Ordonnanceur cosinus avec préchauffage

Le taux d'apprentissage évolue selon un ordonnanceur en deux phases :

i. Phase de préchauffage ($t \leq T_{\text{warmup}}$) Le taux d'apprentissage croît linéairement de zéro à $\eta_{\max}$ durant les premières itérations :

$$\eta(t) = \frac{t}{T_{\text{warmup}}} \cdot \eta_{\max} \tag{3.29}$$

Cette montée progressive stabilise l'entraînement au démarrage, évitant des mises à jour trop agressives avec des poids aléatoires.

ii. Phase de décroissance ($t > T_{\text{warmup}}$) Le taux d'apprentissage décroît selon une fonction cosinus vers $\eta_{\min}$:

$$\eta(t) = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})\left(1 + \cos\left(\frac{t - T_{\text{warmup}}}{T_{\text{total}} - T_{\text{warmup}}}\pi\right)\right) \tag{3.30}$$

Les valeurs utilisées sont :

— $T_{\text{warmup}} = 1000$ pas

— $T_{\text{total}} = 20000$ pas

— $\eta_{\max} = 2 \times 10^{-5}$

— $\eta_{\min} = 2.5 \times 10^{-6}$

Cette stratégie permet d'explorer rapidement l'espace des paramètres durant la phase initiale, puis d'affiner la convergence vers un optimum local en réduisant progressivement le pas de descente.

e) Gradient Clipping

Pour prévenir les explosions de gradients (gradient explosion) susceptibles de déstabiliser l'entraînement, une contrainte de norme est appliquée. Si la norme L2 du gradient global $\|\boldsymbol{g}\|_2$ dépasse un seuil τ, le gradient est rescalé :

$$\tilde{\boldsymbol{g}} = \begin{cases} \boldsymbol{g} & \text{si } \|\boldsymbol{g}\|_2 \leq \tau \\ \frac{\tau}{\|\boldsymbol{g}\|_2} \cdot \boldsymbol{g} & \text{sinon} \end{cases} \tag{3.31}$$

La valeur de seuil utilisée est $\tau = 10.0$ pour tous les modèles. Ce mécanisme assure la stabilité numérique tout en permettant des mises à jour significatives lorsque les gradients restent dans des plages raisonnables.

3.4.3 Génération d'actions par *Flow Matching*

Les architectures GR00T, SmolVLA et Pi0.5 utilisent le *Flow Matching* pour la génération d'actions. Cette méthode probabiliste modélise la génération comme une équation différentielle ordinaire (ODE) permettant de transformer un bruit gaussien initial en une distribution d'actions cohérente, ses paramètres étant optimisés en minimisant l'erreur quadratique moyenne (MSE - Mean Squared Error). Le processus de génération s'effectue par intégration numérique avec un solveur d'Euler.

Le paramètre critique pour l'inférence est le nombre de pas d'intégration K du solveur. Ce paramètre contrôle le compromis entre la précision de la trajectoire générée et la latence computationnelle : une valeur plus élevée améliore la fidélité de la distribution d'actions prédite, mais augmente linéairement le temps de calcul. Pour nos expériences, nous avons fixé $K = 10$ pas d'intégration. Ce choix exploite la propriété des trajectoires rectilignes du *Flow Matching*, qui permettent une convergence rapide avec peu d'itérations, contrairement aux modèles de diffusion classiques nécessitant typiquement 50 à 100 pas.

3.4.4 Métriques de performance en inférence

Pour évaluer objectivement la qualité des politiques apprises, nous utilisons un ensemble de métriques quantitatives mesurant différents aspects des performances opérationnelles.

i. Taux de réussite Cette métrique mesure le pourcentage d'épisodes complétés avec succès lors de l'exécution autonome sur le robot, reflétant l'utilité pratique de la politique apprise. L'évaluation est réalisée en boucle fermée (cf. Section a)) jusqu'à l'accomplissement de la tâche ou l'atteinte d'un critère d'échec temporel : si la durée d'exécution dépasse un seuil fixé sans progression observable, l'épisode est interrompu et marqué comme échec.

Il convient de souligner un avantage observé durant nos expérimentations : les modèles VLA peuvent exhiber des comportements adaptatifs non explicitement présents dans les données d'entraînement. Nous observons par exemple plusieurs tentatives de saisie lorsque la première échoue, alors qu'aucune démonstration d'entraînement ne contenait de trajectoires incluant des échecs suivis de nouvelles tentatives. Ce comportement émergent illustre la capacité des politiques génératives à improviser des stratégies de récupération à partir de leur compréhension générale de la tâche. Une analyse quantitative des taux de récupération sur 5 séries d'évaluations (soit 500 essais cumulés) est présentée en annexe (Section A.4.2).

ii. Latence d'inférence La latence d'inférence comprend trois composantes : (i) acquisition et prétraitement des observations multimodales (capteurs, redimensionnement et normalisation des images), (ii) inférence du modèle VLA (incluant les $K = 10$ pas d'intégration du *Flow Matching*), et (iii) post-traitement de dénormalisation des actions. Pour maintenir une exécution temps-réel avec la

segmentation d'actions (Section a)), la latence totale doit rester inférieure à la durée d'un segment.

3.5 Évaluation et comparaison des architectures

Cette section présente l'évaluation expérimentale des architectures VLA sur des tâches de manipulation robotique réelles. Nous commençons par caractériser les trois jeux de données collectés, puis décrivons le protocole de test standardisé permettant une comparaison objective des performances. L'analyse comparative qui suit quantifie l'impact de chaque composante (architecture, horizon de prédiction, taille de lot) sur le taux de réussite et l'efficacité d'exécution.

3.5.1 Analyse des trois jeux de données collectés

Afin d'évaluer les performances des architectures VLA sur des tâches de manipulation variées, nous avons collecté trois jeux de données adaptés aux capacités physiques du bras robotique SO-101 et à son espace de travail.

Le dataset **pick-place-red-block** servira de benchmark de référence pour l'évaluation comparative des politiques, en vue de sélectionner l'architecture la plus performante. Cette tâche atomique consiste à saisir un bloc puis à le déposer dans un récipient, ce qui représente l'opération fondamentale de pick-and-place, omniprésente dans les chaînes de production industrielles.

Le dataset **sort-blocks** consiste à trier des blocs de couleurs différentes en plaçant les blocs bleus dans un conteneur bleu et les blocs verts dans un conteneur blanc. Cette tâche est analogue aux opérations de tri industriel, notamment l'inspection qualité où le tri est basé sur des critères visuels tels que les défauts de surface ou autres variations.

Le dataset **arrange-table** simule une tâche de rangement de plusieurs objets cylindriques (stylos) dispersés dans un conteneur, une tâche nécessitant plus de dextérité et de précision.

Le tableau 3.5 synthétise les caractéristiques quantitatives de ces trois jeux de données.

La figure 3.2 présente l'étendue des articulations pour chaque dataset, définie comme la différence entre les positions normalisées maximale et minimale observées lors de l'exécution des trajectoires. Cette métrique quantifie la diversité des configurations articulaires utilisées pour accomplir chaque tâche : une faible étendue signifie que l'articulation reste dans une zone restreinte de sa plage, tandis qu'une

TABLEAU 3.5 – Caractéristiques des jeux de données d'entraînement

Jeu de données	Nombre Épisodes	Nombre Total Frames	Nombre Moy. Frames	Nombre Max Frames	Durée Moy. (s)	FPS
arrange-table	73	60 609	830	1305	27.7	30
pick-place-red-block	239	31 234	131	243	4.4	30
sort-blocks	43	52 936	1231	1574	41.0	30

étendue élevée indique que l'articulation doit explorer largement sa capacité de mouvement.

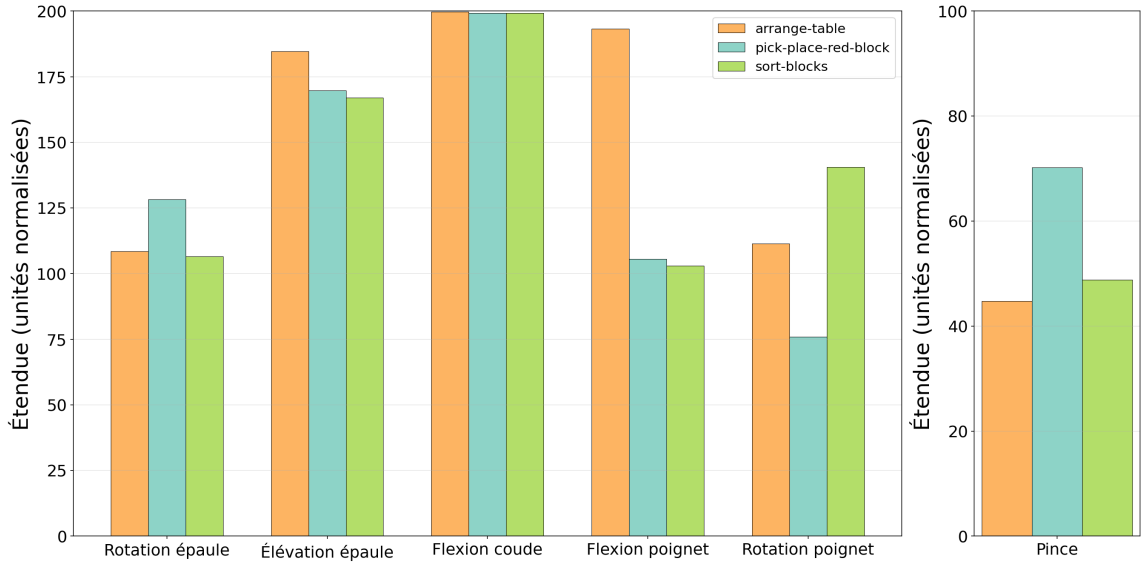


FIGURE 3.2 – Statistiques des articulations par jeu de données

La figure 3.2 présente l'étendue des articulations pour chaque dataset. À partir de ces données et du tableau 3.5, on observe que `pick-place-red-block` correspond à des épisodes courts centrés sur une tâche atomique, tandis que `sort-blocks` et `arrange-table` contiennent des épisodes nettement plus longs couvrant la manipulation de plusieurs objets.

Les articulations flexion du coude et élévation de l'épaule présentent les plus fortes étendues (proches de 200 et de 170-185 unités normalisées selon les datasets), ce qui indique qu'elles doivent explorer largement leur capacité de mouvement, indépendamment de la tâche exécutée. Cette sollicitation reflète leur rôle central pour étendre ou rétracter le bras et ajuster sa hauteur d'opération.

L'articulation rotation en lacet de l'épaule présente des étendues comprises entre 106 et 127 unités normalisées selon les datasets, indiquant qu'elle reste dans une zone restreinte de sa plage, suffisante pour orienter le bras sans rotations extrêmes.

L'articulation flexion du poignet présente une étendue de 193.2 unités pour `arrange-table`, nettement supérieure aux valeurs de 105.6 et 102.9 observées pour `pick-place-red-block` et `sort-blocks` respectivement. Cette différence caractérise la nature de la tâche de rangement de stylos, nécessitant des inclinaisons variées du poignet pour manipuler des objets cylindriques fins à différentes orientations.

L'étendue de la rotation en roulis du poignet atteint 140.5 unités pour `sort-blocks`, valeur supérieure aux 111.4 et 75.9 observées pour `arrange-table` et `pick-place-red-block`. Cette différence traduit la nécessité de rotations du poignet plus prononcées lors du tri de blocs vers des conteneurs distincts positionnés à différents endroits de l'espace de travail.

Concernant la pince, l'étendue de 70.2 unités pour `pick-place-red-block` est supérieure à celle de `sort-blocks` (48.8) et `arrange-table` (44.8). Cette hiérarchie correspond directement aux dimensions des objets manipulés : le bloc rouge constitue l'objet le plus volumineux, suivi des blocs de tri de taille moyenne, puis des stylos fins requérant une ouverture réduite de la pince.

3.5.2 Environnement de test standardisé

Afin de comparer objectivement les performances des trois architectures VLA entraînées, nous avons mis en place un environnement de test standardisé basé sur la tâche atomique de *pick-and-place*. Ce protocole d'évaluation garantit des conditions identiques pour chaque politique, permettant une comparaison équitable des taux de réussite et des temps d'exécution.

a) Génération des positions de test

L'évaluation repose sur un ensemble de 100 configurations prédéfinies, chacune spécifiant les positions du bloc rouge et du récipient dans l'espace de travail. Ces positions sont générées aléatoirement dans une zone valide correspondant à la portée du bras SO-101. Comme illustré dans la figure 3.3, l'espace de travail est divisé en deux régions : une zone valide (en vert) couvrant la majorité de la surface accessible, et une zone d'exclusion (en rouge) correspondant à la position physique du bras robotique. La pré-génération assure une distribution uniforme, permet la reproductibilité exacte des expériences et élimine tout biais lié au choix manuel des positions.

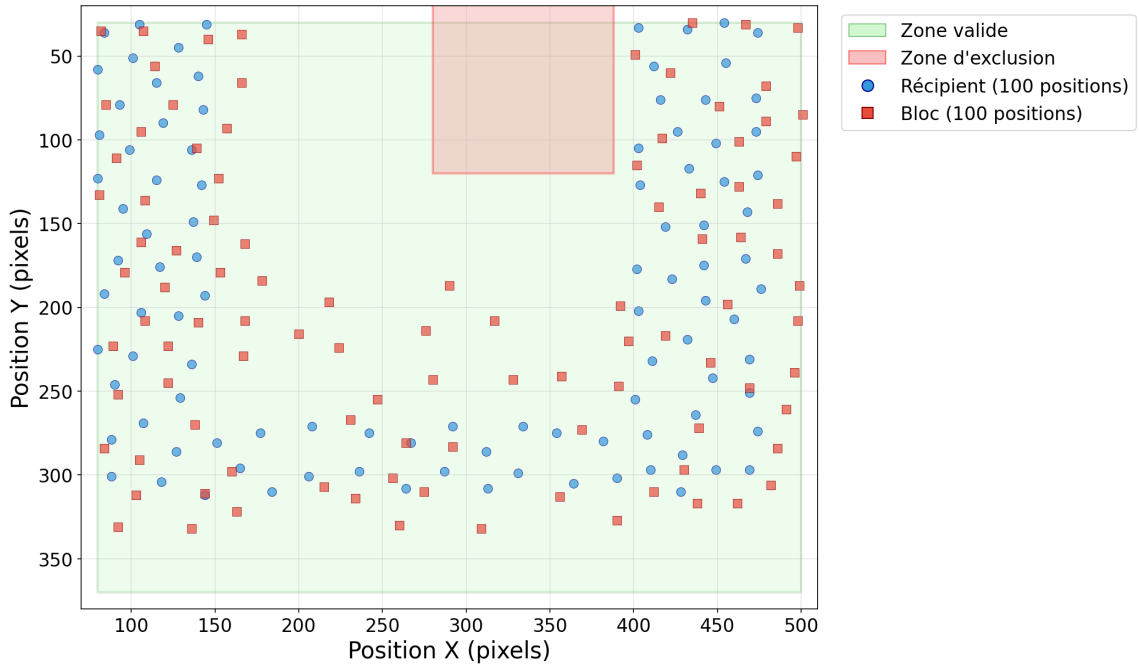
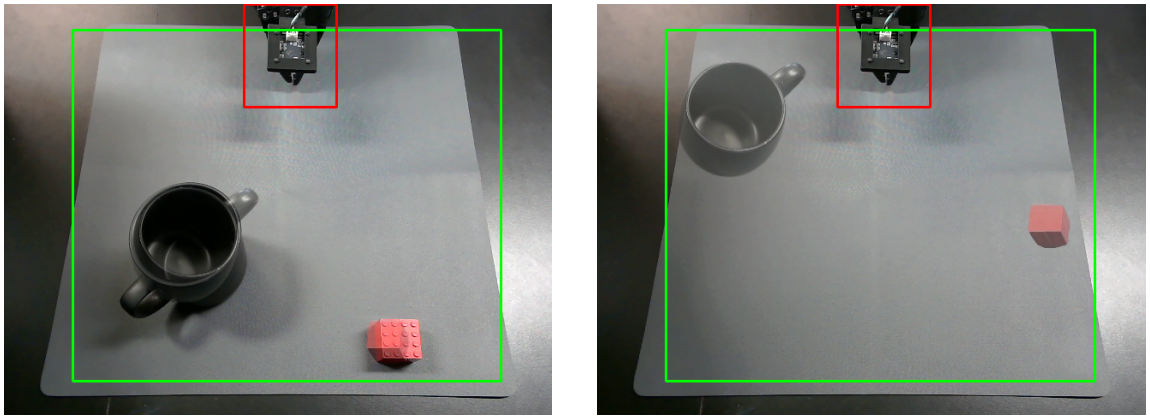


FIGURE 3.3 – Distribution des 100 positions de test générées.

b) Protocole d'exécution

Pour chaque épisode de test, l'opérateur positionne le bloc et le récipient selon les coordonnées spécifiées, en s'appuyant sur un repère visuel superposé au flux de la caméra frontale, comme l'illustre la figure 3.4.



(a) Configuration 1 (b) Configuration 2

FIGURE 3.4 – Exemples de repères visuels de positionnement utilisés pour standardiser le placement des objets avant chaque épisode de test.

Une fois l'environnement configuré, le bras est initialisé dans sa position de repos (*home base*) et l'inférence démarre. La politique reçoit les observations visuelles

des deux caméras ainsi que l'instruction textuelle, puis génère en boucle fermée les actions motrices jusqu'à l'accomplissement de la tâche ou l'atteinte d'un critère d'échec.

c) Classification du succès

À la fin de chaque épisode, le système enregistre le temps d'inférence total et l'opérateur indique manuellement si la tâche a été accomplie avec succès ou non. Pour automatiser cette étape, nous avons entraîné un modèle de classification d'images basé sur YOLOv12. Ce classifieur prend en entrée la dernière frame capturée par la caméra frontale et prédit la classe (succès ou échec), accélérant le processus d'évaluation.

Les résultats de chaque épisode (numéro de position, succès/échec, temps d'exécution, prédiction YOLO et confiance) sont consignés dans un fichier CSV, permettant une analyse statistique ultérieure des performances de chaque architecture.

3.6 Résultats comparatifs et analyse des performances

Cette section présente une analyse des performances des différents modèles VLA entraînés. Notre évaluation comprend cinq axes pour isoler l'impact de chaque module : une comparaison directe des architectures à hyperparamètres constants, une étude de l'influence de l'horizon de prédiction (*chunk size*), une analyse de l'effet de la taille de lot (*batch size*), une expérience de distillation de connaissances pour améliorer les petits modèles, et enfin une validation des techniques de lissage pour la fluidité des mouvements.

3.6.1 Comparaison des architectures

Nous commençons par comparer les performances intrinsèques des trois architectures retenues (Pi0.5, SmolVLA et GR00T) en les entraînant dans des conditions strictement identiques. Cette comparaison de référence permet d'établir la hiérarchie des modèles sur la tâche de manipulation standard.

a) Métriques d'entraînement

La figure 3.5 présente l'évolution du taux d'apprentissage et de la perte durant l'entraînement des trois architectures. Les profils de taux d'apprentissage (*learning*

rate) révèlent des stratégies d'optimisation distinctes : SmolVLA et GR00T adoptent un ordonnanceur cosinus avec une phase de préchauffage (*warm-up*) rapide atteignant un pic à 10^{-4} avant une décroissance progressive, tandis que Pi0.5 utilise un taux d'apprentissage significativement plus faible (2.5×10^{-5}) qui se stabilise après 3000 étapes.

L'analyse des courbes de perte met en évidence une convergence nettement plus rapide pour GR00T et SmolVLA, atteignant des valeurs inférieures à 0.02 après 5000 étapes. Pi0.5 présente une perte initiale élevée (0.35) qui décroît graduellement mais se stabilise autour de 0.05, soit environ cinq fois supérieure aux deux autres modèles.

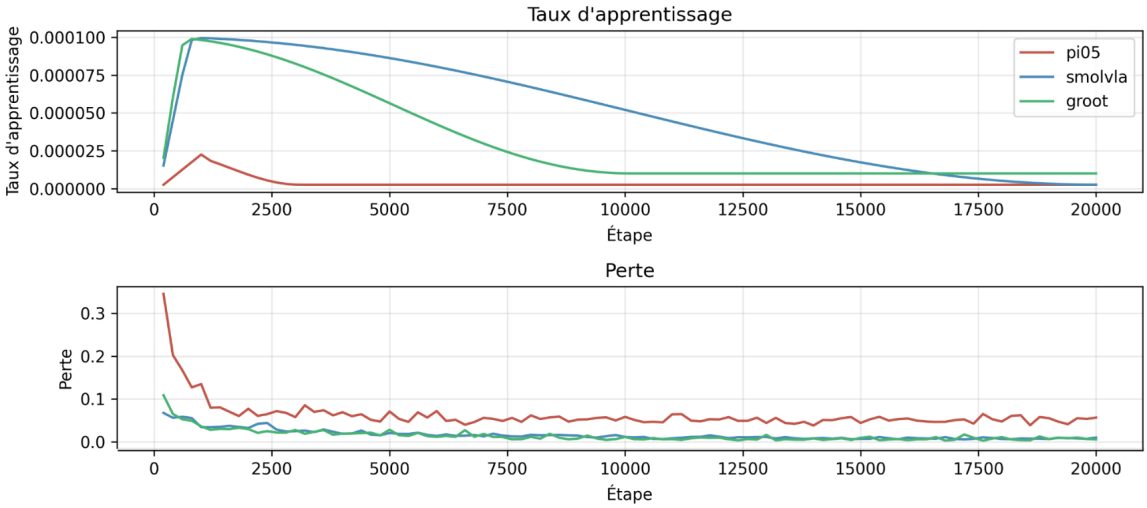


FIGURE 3.5 – Évolution du taux d'apprentissage et de la perte pour les trois architectures (Pi0.5, SmolVLA, GR00T)

La figure 3.6 compare les ressources computationnelles requises par chaque architecture. SmolVLA se distingue par son efficacité remarquable avec un temps total par étape de 0.85 secondes, contre 2.01 secondes pour GR00T et 2.84 secondes pour Pi0.5. Cette efficacité provient de sa taille réduite (428M paramètres) comparée à GR00T (3B) et Pi0.5 (4B).

L'empreinte mémoire GPU reflète directement cette hiérarchie de taille : SmolVLA requiert seulement 22.4 Go de VRAM, tandis que GR00T et Pi0.5 nécessitent respectivement 43.7 Go et 44.6 Go. La décomposition du temps par étape révèle des profils distincts : Pi0.5 consacre la quasi-totalité de son temps à la mise à jour des poids (2.79s sur 2.84s), alors que GR00T présente un temps de chargement des données prédominant (1.49s sur 2.01s), suggérant que l'optimisation du pipeline de données pourrait améliorer significativement ses performances d'entraînement.

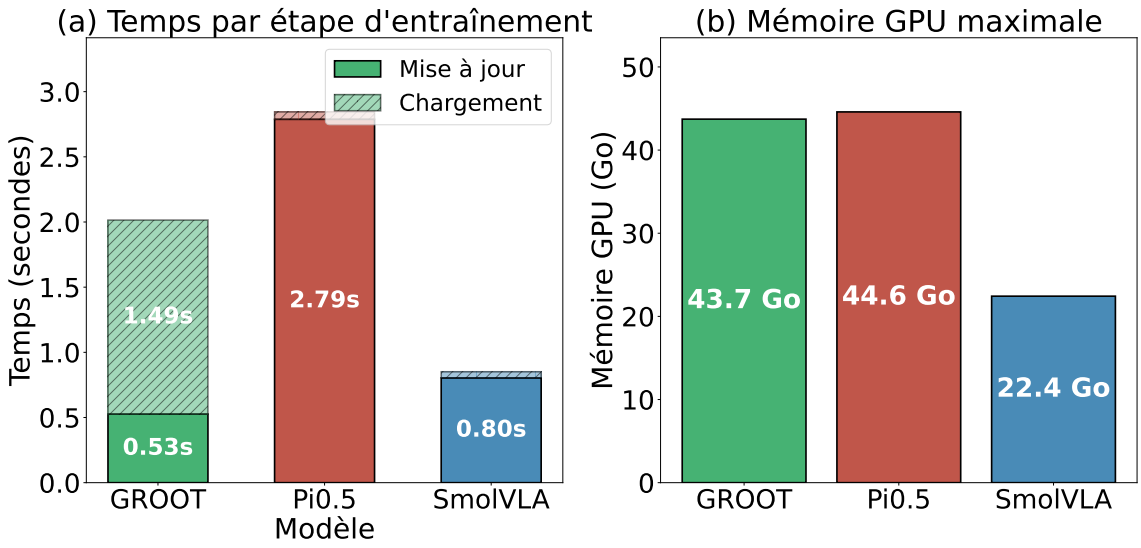


FIGURE 3.6 – Comparaison des ressources d'entraînement entre les modèles

b) Métriques d'inférence

Nous évaluons maintenant les performances opérationnelles des trois architectures sur l'environnement de test standardisé comprenant 100 configurations spatiales distinctes, et la figure 3.7 synthétise ces résultats.

L'analyse des performances d'inférence révèle une hiérarchie nette entre les trois architectures. GR00T atteint un taux de succès de 93%, surpassant significativement Pi0.5 (62%) et SmolVLA (61%). Cette supériorité s'accompagne d'une efficacité trajectoire optimale : GR00T accomplit la tâche avec une moyenne de 171.6 actions ($\sigma = 67.9$), contre 216.1 actions ($\sigma = 69.4$) pour SmolVLA et 313.5 actions ($\sigma = 182.3$) pour Pi0.5. Ces métriques démontrent que GR00T génère des trajectoires plus directes et déterministes, minimisant les mouvements redondants et les corrections superflues. À l'inverse, Pi0.5 exhibe un comportement plus erratique avec un écart-type près de trois fois supérieur, traduisant une variabilité importante dans la qualité des prédictions et la présence de mouvements parasites. Le tableau A.2, disponible en Annexe complète, détaille les statistiques par série.

La latence d'inférence moyenne reflète ces différences : GR00T présente un temps d'inférence de 7.46 secondes par épisode réussi, SmolVLA de 9.79 secondes, et Pi0.5 de 14.86 secondes, soit le double de GR00T. Cette latence accrue pour Pi0.5 résulte directement du nombre supérieur d'actions nécessaires pour converger vers l'objectif.

La répartition des causes d'échec met en évidence la difficulté intrinsèque de la phase de préhension. Pour SmolVLA, 84.6% des échecs surviennent lors de la saisie du bloc contre 15.4% lors du dépôt dans le conteneur. Pi0.5 présente une distribution

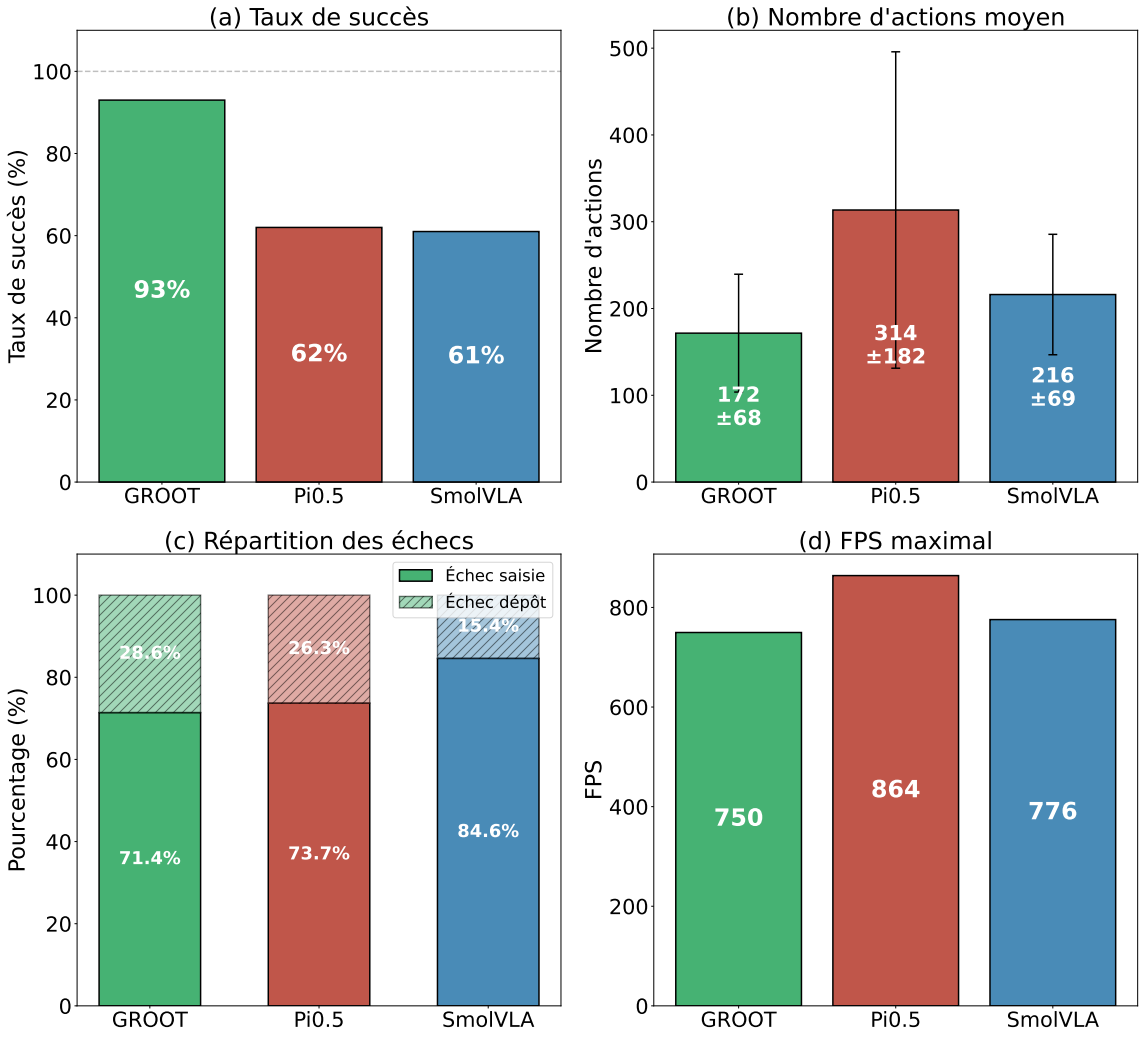


FIGURE 3.7 – Comparaison des performances d'inférence entre les modèles GR00T, Pi0.5 et SmolVLA

similaire avec 73.7% d'échecs à la préhension et 26.3% au dépôt. Cette prédominance des échecs à la saisie s'explique par les dimensions réduites du bloc rouge qui exigent une précision millimétrique des prédictions articulaires, particulièrement aux positions extrêmes de l'espace de travail où les erreurs angulaires cumulées sur les six articulations amplifient l'écart entre la position prédite et la position cible du préhenseur.

Malgré la taille considérable des modèles, GR00T compte 3 milliards de paramètres et Pi0.5 environ 4 milliards, les débits d'inférence mesurés atteignent respectivement 749.67 FPS et 863.95 FPS, des performances comparables aux 775.6 FPS de SmolVLA. Ces résultats démontrent que la taille du modèle n'est pas le seul facteur déterminant, grâce à deux techniques d'optimisation complémentaires :

— **Précision mixte (BFloat16/FP32)** : Les opérations matricielles intensives (multiplications matricielles dans les couches d'attention et les réseaux de neurones à propagation avant **FFN**, *Feed-Forward Network*) sont effectuées en précision réduite **BFloat16** (*Brain virgule flottante 16 bits*), réduisant de moitié l'empreinte mémoire et accélérant les calculs. Les poids du modèle et les gradients accumulés sont maintenus en **FP32** (*virgule flottante 32 bits*) pour préserver la stabilité numérique lors des mises à jour. Cette technique est utilisée par Pi0.5 et GR00T, tandis que SmolVLA opère en précision FP32 complète.

— **Compilation JIT avec `torch.compile()`** : Pi0.5 bénéficie de la compilation **JIT** (*Just-In-Time*, compilation à la volée) de PyTorch 2.0. Contrairement à l'exécution standard qui interprète les opérations une par une, la compilation JIT analyse le graphe de calcul complet lors du premier passage et le transforme en code machine optimisé. Après ce délai initial de compilation, les passages suivants s'exécutent beaucoup plus rapidement grâce aux optimisations appliquées (fusion d'opérations, élimination de synchronisations redondantes). Cette optimisation, utilisée uniquement par Pi0.5, explique son débit maximal supérieur (863.95 FPS) par rapport à GR00T (749.67 FPS) malgré sa taille supérieure.

Bien que ces débits théoriques soient élevés, l'inférence réelle est manuellement limitée à 30 Hz pour garantir des mouvements fluides et contrôlés. Ce délai d'attente inter-actions permet au bras robotique de compléter le déplacement prédit avant l'application de la commande suivante. Des expériences complémentaires d'évaluation, totalisant 5 séries de 100 essais pour chaque architecture, confirment ces tendances et sont détaillées en annexe (Section A.4.1).

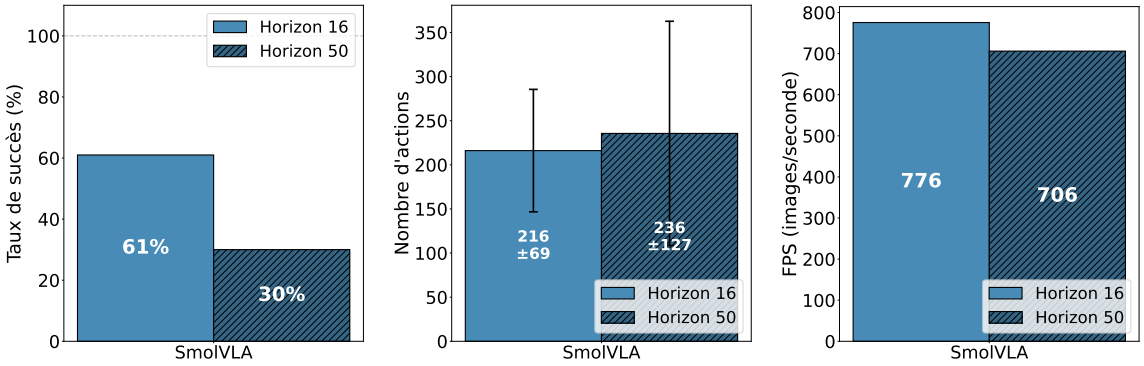
3.6.2 Impact de l'horizon de prédiction (*Chunk Size*)

L'horizon de prédiction détermine le nombre d'actions futures générées par le modèle à chaque inférence. Nous analysons ici comment l'extension de cet horizon de 16 à 50 pas affecte la précision et la cohérence temporelle du modèle SmolVLA.

a) Métriques d'inférence

Pour évaluer l'impact de l'horizon de prédiction sur les performances opérationnelles, nous avons déployé les deux versions du modèle SmolVLA (horizon

de 16 et 50 pas) sur l'environnement de test standardisé. La figure 3.8 présente les métriques clés de cette évaluation.



(a) Taux de succès (b) Nombre d'actions (c) FPS maximal

FIGURE 3.8 – Impact de l'horizon de prédiction sur les performances d'inférence

L'analyse des performances d'inférence révèle une dégradation significative lors de l'extension de l'horizon de prédiction de 16 à 50 pas temporels. Comme l'illustre la figure 3.8a, le taux de succès chute drastiquement de 61% à 30%, indiquant que la prédiction de séquences d'actions plus longues compromet la capacité du modèle à accomplir la tâche de manipulation.

Le nombre moyen d'actions nécessaires pour compléter un épisode augmente légèrement, passant de 216.1 ($\sigma = 69.4$) à 235.6 ($\sigma = 127.1$) actions, tel que présenté dans la figure 3.8b. Cette augmentation, bien que modeste en valeur absolue, s'accompagne d'un quasi-doublement de l'écart-type, traduisant une variabilité accrue dans l'exécution des trajectoires. Les épisodes réussis avec un horizon de prédiction de 50 nécessitent davantage de corrections et présentent des comportements moins déterministes.

Le débit d'inférence maximal, illustré par la figure 3.8c, diminue de 775.6 FPS à 706.1 FPS, soit une réduction de 9%. Cette baisse de performance s'explique par l'augmentation de la dimension de sortie du modèle : la génération d'une matrice d'actions de dimension 50×6 (300 valeurs) requiert davantage de calculs que la prédiction d'une matrice 16×6 (96 valeurs). Bien que le modèle effectue moins d'appels d'inférence (une nouvelle génération tous les 50 pas contre 16), le coût computationnel par inférence augmente de manière non linéaire avec l'horizon de prédiction, annulant partiellement cet avantage.

Ces résultats s'expliquent par l'équilibre entre réactivité et efficacité d'exécution. Avec un horizon de 16 pas, le modèle ré-observe l'environnement plus fréquemment, lui permettant d'adapter ses prédictions à la scène visuelle courante. Un horizon

de 50 pas espace davantage ces mises à jour, réduisant la fréquence de correction. Or, à mesure que le robot exécute les actions d'un bloc, sa configuration change et la scène évolue : plus le modèle ré-observe fréquemment son état réel, plus ses prédictions restent précises. Avec un horizon de 50, les dernières actions du bloc sont conditionnées sur une observation qui s'éloigne progressivement de la scène courante, ce qui contribue à la baisse du taux de réussite observée.

Ces observations sont cohérentes avec les résultats rapportés par les auteurs de SmolVLA (Shukor et coll. 2025), qui montrent que les valeurs extrêmes de l'horizon de prédiction (trop petites ou trop grandes) dégradent les performances, et que la plage optimale se situe entre 10 et 50 pas. Notre configuration $C = 16$ se place dans cette zone optimale. De plus, les auteurs soulignent qu'exécuter la totalité du bloc d'actions avant de mettre à jour les observations accélère l'inférence, mais réduit la capacité du robot à réagir aux changements de l'environnement. Ce compromis entre vitesse d'inférence et précision de contrôle confirme l'analyse théorique présentée en Section a) : un horizon court favorise une correction de trajectoire plus fréquente, ce qui est critique pour les tâches de manipulation fine où les perturbations mineures peuvent compromettre la réussite de l'épisode. Les résultats détaillés de cette validation sur 5 séries d'évaluations sont présentés en annexe dans la Section A.4.3 ainsi que dans le tableau A.3.

3.6.3 Influence de la taille de lot (*Batch Size*)

La taille de lot est un hyperparamètre important pour la stabilité de l'apprentissage, en particulier pour les modèles basés sur les transformeurs. Nous étudions l'impact de l'augmentation de la taille de lot de 64 à 120 sur les performances du modèle GR00T.

a) Métriques d'entraînement

Pour isoler l'effet de la taille de lot sur la généralisation, nous avons ajusté le nombre d'étapes d'entraînement afin de maintenir un nombre comparable d'échantillons vus par le modèle. Une *époque* (epoch) correspond au nombre de passages complets sur l'ensemble des données d'entraînement, donc le nombre de fois où le modèle observe chaque échantillon du dataset exactement une fois.

Le modèle de référence avec une taille de lot de 64 a été entraîné pendant 20 000 étapes, soit environ 40.98 époques, pour un total de 1 280 000 échantillons vus. En comparaison, le modèle avec une taille de lot de 120 a été entraîné pendant seulement

10 000 étapes, correspondant à environ 38.42 époques et 1 200 000 échantillons vus. Cette configuration permet une comparaison équitable où les deux modèles ont été exposés à une quantité similaire de données d'entraînement. Dans la suite, nous comparons les configurations utilisant une taille de lot de 64 et de 120.

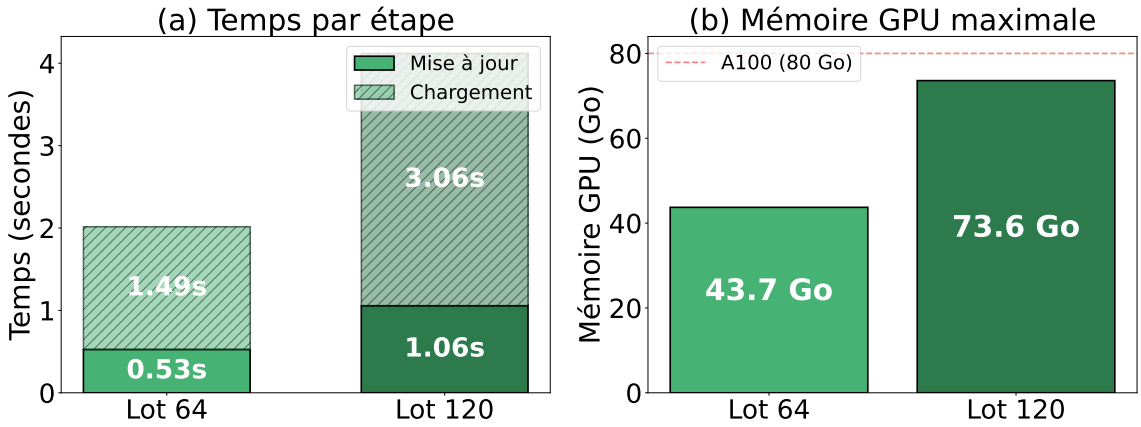


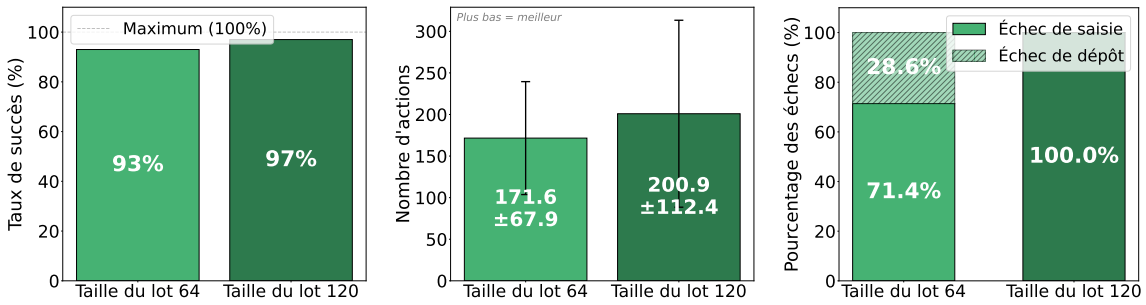
FIGURE 3.9 – Impact de la taille du lot sur les ressources d'entraînement (GROOT)

L'analyse des ressources d'entraînement, illustrée à la figure 3.9, révèle le compromis inhérent à l'augmentation de la taille de lot. Le temps par étape double approximativement, passant de 2.01 secondes (0.53s de mise à jour + 1.49s de chargement) pour la taille de lot de 64 à 4.12 secondes (1.06s + 3.06s) pour la taille de lot de 120. La consommation mémoire GPU augmente aussi de 43.7 Go à 73.6 Go, approchant la limite de 80 Go de notre carte A100, mais on observe que le modèle avec une taille de lot de 120 converge en deux fois moins d'étapes, résultant en un temps d'entraînement total comparable.

b) Métriques d'inférence

Nous testons maintenant l'impact de la taille de lot sur les performances réelles du modèle lors du déploiement sur le robot.

Les résultats d'inférence démontrent que l'augmentation de la taille de lot améliore la généralisation du modèle. Comme illustré par la figure 3.10a, le taux de succès progresse de 93% à 97%, soit une amélioration de 4 points de pourcentage. Cette amélioration s'accompagne d'une augmentation du nombre moyen d'actions par épisode de 171.6 ($\sigma = 67.9$) à 200.9 ($\sigma = 112.4$), phénomène mis en évidence dans la figure 3.10b. Cette légère augmentation, accompagnée d'un écart-type plus élevé, est potentiellement liée à une meilleure généralisation grâce au nombre plus élevé



(a) Taux de succès (b) Nombre d'actions (c) Raisons d'échec

FIGURE 3.10 – Impact de la taille du lot sur les performances d'inférence (GROOT)

d'observations vues avant la mise à jour des poids à chaque étape d'entraînement, permettant des trajectoires plus exploratoires.

L'analyse des raisons d'échec, présentée dans la figure 3.10c, montre que pour la taille de lot de 64, sur les 7 échecs observés, 5 surviennent lors de la saisie du bloc (71.4%) et 2 lors du dépôt dans le récipient (28.6%). En revanche, pour la taille de lot de 120, les 3 échecs proviennent exclusivement de la phase de saisie : une fois le bloc saisi, le modèle réussit systématiquement à le déposer dans le récipient. Ces échecs lors de la phase de préhension s'expliquent par le fait que cette phase exige une précision spatiale plus élevée pour positionner correctement la pince. Le tableau A.4 présenté en Annexe complète fournit les résultats détaillés sur 5 séries.

Cette amélioration de la généralisation s'explique par l'estimation plus stable du gradient permise par les lots plus grands. Avec 120 échantillons par lot au lieu de 64, la variance du gradient estimé diminue d'un facteur $\sqrt{120/64} \approx 1.37$, permettant des mises à jour de poids plus cohérentes et une convergence vers des minima plus généralisables. De plus, les lots plus grands offrent une meilleure représentation de la diversité des démonstrations à chaque étape, réduisant le risque de surapprentissage sur des configurations particulières du dataset. Les résultats détaillés de cette validation sur 5 séries d'évaluations sont présentés en annexe (Section A.4.4).

3.6.4 Amélioration par distillation de connaissances

Afin de combiner la légèreté de SmolVLA avec la performance de modèles plus complexes, nous explorons une approche de distillation de connaissances. L'objectif est d'améliorer le modèle "élève" SmolVLA (428M paramètres) en exploitant les connaissances acquises par le modèle "enseignant" GR00T, plus performant mais également plus coûteux en ressources computationnelles.

La distillation de connaissances est une technique d'apprentissage par transfert où un modèle compact (élève) apprend à reproduire le comportement d'un modèle plus large (enseignant). Contrairement à la distillation classique qui minimise la divergence entre les distributions de sortie des deux modèles, nous adoptons ici une approche par imitation : le modèle enseignant génère des trajectoires de démonstration que le modèle élève apprend ensuite à reproduire.

Pour constituer l'ensemble de données de distillation, nous avons déployé le modèle GR00T pré-entraîné (points de contrôle taille de lot de 120 et 64, et l'horizon de prédiction de 16 actions) sur la tâche de *pick-and-place* et enregistré les trajectoires réussies. Au total, 199 épisodes réussis ont été collectés, représentant 27 904 frames d'observations et d'actions. L'avantage de cette approche réside dans l'élimination de la collecte manuelle de nouvelles trajectoires par le téléopérateur, une tâche coûteuse en effort et en temps. Cette méthode permet ainsi une amélioration continue des performances du modèle élève sans intervention humaine supplémentaire.

a) Métriques d'entraînement

L'ajustement fin (*fine-tuning*) du modèle SmolVLA est réalisé à partir du point de contrôle précédemment entraîné sur les données de téléopération humaine (20 000 étapes). Pour éviter le surapprentissage sur l'ensemble de données de distillation, nous limitons l'entraînement à 2 000 étapes supplémentaires avec un taux d'apprentissage réduit. Avec une taille du lot de 64 et 27 904 frames disponibles, chaque époque correspond à environ 436 étapes, soit environ 4.59 époques sur l'ensemble de l'ajustement fin.

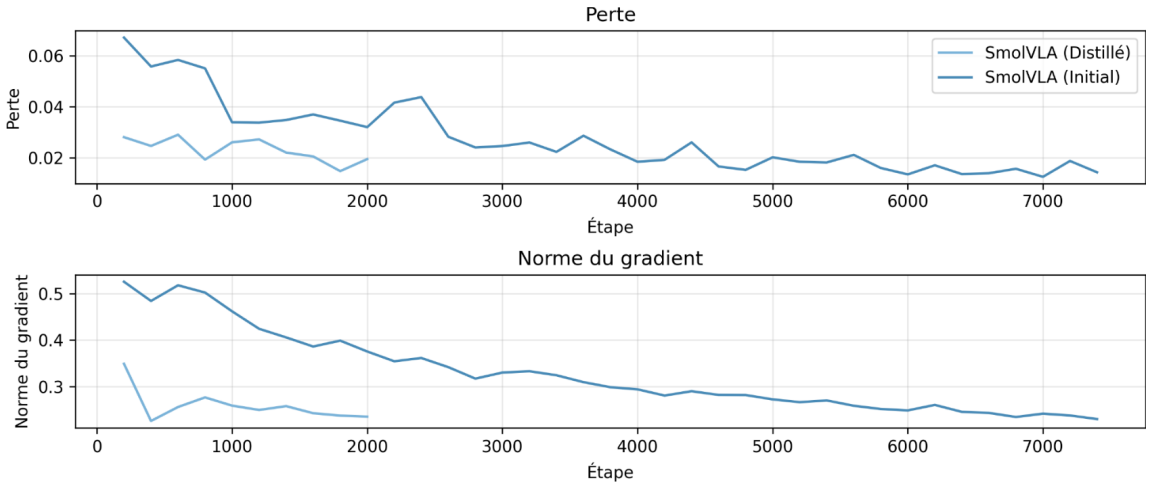


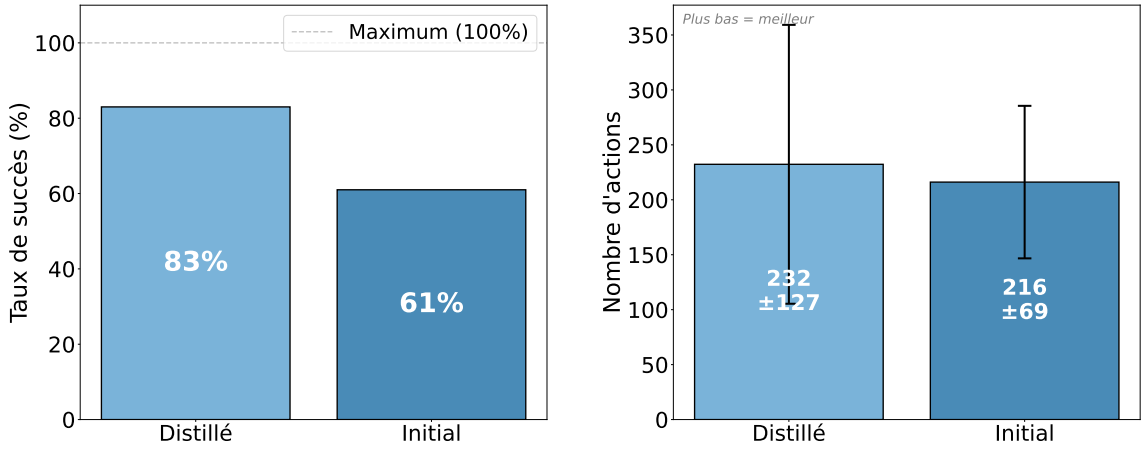
FIGURE 3.11 – Comparaison de la convergence entre le modèle initial et le modèle distillé (Perte et Norme du gradient)

La figure 3.11 compare l'évolution de la perte et de la norme du gradient entre le modèle initial (entraîné depuis les poids pré-entraînés) et le modèle distillé (ajusté à partir du point de contrôle déjà optimisé sur les données humaines). Le modèle distillé démarre avec une perte significativement plus basse (environ 0.024 contre 0.067 à l'étape 200), ce qui s'explique par le fait qu'il bénéficie des poids déjà optimisés lors de la première phase d'ajustement fin sur les données de téléopération. Cette initialisation avantageuse permet au modèle de partir d'un point plus proche de l'optimum, réduisant ainsi la perte initiale et accélérant la convergence.

La norme du gradient présente également un comportement distinct : le modèle distillé affiche des valeurs plus faibles (0.24-0.35 contre 0.38-0.52), témoignant d'une surface de perte plus lisse et de mises à jour de poids plus stables.

b) Métriques d'inférence

Nous évaluons maintenant si les améliorations observées durant l'entraînement se traduisent par de meilleures performances lors du déploiement réel sur le robot.



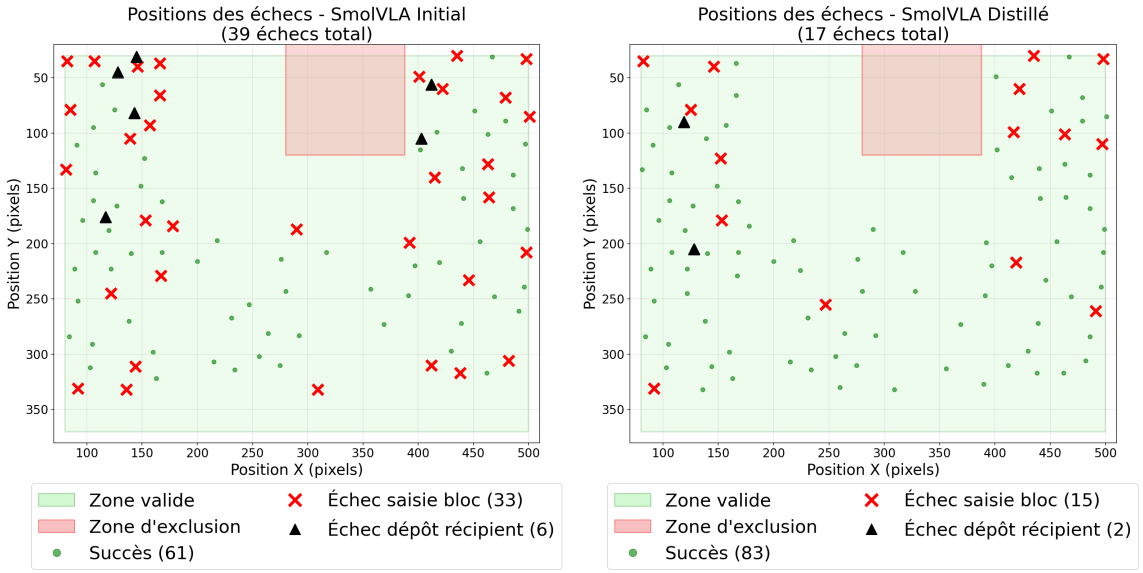
(a) Taux de succès (b) Nombre d'actions

FIGURE 3.12 – Impact de la distillation sur les performances d'inférence (SmolVLA)

Les résultats d'inférence démontrent l'efficacité de l'approche de distillation. Comme l'illustre la figure 3.12a, le taux de succès progresse de 61% à 83%, soit une amélioration de 22 points de pourcentage. Cette amélioration substantielle confirme que le modèle élève a effectivement acquis des comportements plus robustes en apprenant des trajectoires optimales du modèle enseignant. Le tableau A.5 en Annexe complète fournit les statistiques détaillées sur 5 séries.

L'analyse du nombre d'actions par épisode, présentée dans la figure 3.12b, révèle un transfert comportemental intéressant. Le modèle SmolVLA initial accomplissait la

tâche en 216.1 actions ($\sigma = 69.4$), contre 200.9 actions ($\sigma = 112.4$) pour l'enseignant GR00T. Après distillation, le modèle élève présente 232.3 actions avec un écart-type de 126.9, se rapprochant du profil de l'enseignant. Cette variabilité accrue traduit l'acquisition de stratégies plus diversifiées, caractéristiques de la robustesse du modèle enseignant.



(a) SmolVLA Initial (b) SmolVLA Distillé

FIGURE 3.13 – Comparaison des positions d'échec entre le modèle initial et le modèle distillé

La figure 3.13 présente la distribution spatiale des échecs pour les deux modèles. En examinant la figure 3.13a pour le modèle initial, on observe 39 échecs au total, dont 33 erreurs de saisie du bloc (croix rouges) et 6 erreurs de dépôt dans le récipient (croix noires). Ces échecs se concentrent principalement dans les coins de l'espace de travail, particulièrement dans les zones proches du bras robotique où la préhension requiert une coordination précise de l'ensemble des articulations.

En comparaison, le modèle distillé, présenté à la figure 3.13b, montre une réduction significative du nombre total d'échecs, passant de 39 à 17 (15 erreurs de saisie et 2 erreurs de dépôt). La distribution spatiale des échecs reste similaire, avec une concentration dans les coins et les bords de l'espace de travail. Ces zones correspondent aux configurations articulaires limites où le bras atteint ses butées mécaniques, rendant les mouvements fins particulièrement difficiles. Néanmoins, le nombre d'échecs dans ces zones critiques est considérablement réduit, démontrant que la distillation a permis au modèle d'acquérir des stratégies de manipulation plus précises, même dans les configurations géométriquement contraintes. Les résultats

détaillés rapportés sur 5 séries d'évaluations de 100 essais corroborent ces conclusions et sont présentés en annexe (Section A.4.5).

3.6.5 Optimisation de la fluidité et réduction des micro-tremblements

Enfin, nous abordons la qualité du mouvement robotique en évaluant l'efficacité des techniques de pré-traitement des données visant à réduire les micro-tremblements (*jitter*). Cette étude est menée sur la tâche de tri de blocs avec le modèle GR00T, en comparant l'apprentissage sur données brutes aux données lissées.

L'analyse des trajectoires générées par les politiques VLA révèle la présence de micro-tremblements : des mouvements redondants où le robot maintient une position quasi-stationnaire pendant plusieurs frames consécutives. À une fréquence d'échantillonnage de 30 Hz, les micro-mouvements de la main de l'opérateur humain sont enregistrés comme des frames quasi-statiques et le modèle, entraîné par apprentissage par imitation, apprend à reproduire ces redondances lors de l'inférence.

Formellement, soit une séquence de vecteurs d'actions $\mathbf{a}_t = (a_t^{(1)}, \ldots, a_t^{(J)}) \in \mathbb{R}^J$ représentant les $J = 6$ positions articulaires normalisées à chaque pas de temps t. L'algorithme de filtrage maintient une **frame de référence** (ancre) correspondant à la dernière frame retenue, d'indice t_{ref}, initialisée à la première frame de l'épisode ($t_{\text{ref}} = 0$). Chaque frame successive $t > t_{\text{ref}}$ est comparée à cette ancre, et non à la frame immédiatement précédente $t-1$. La frame t est classifiée comme redondante si le déplacement maximal sur l'ensemble des articulations par rapport à l'ancre reste inférieur à un seuil δ :

$$\max_{j=1}^{J} \left| a_t^{(j)} - a_{t_{\text{ref}}}^{(j)} \right| < \delta \tag{3.32}$$

Tant que cette condition est satisfaite, la frame t est supprimée et l'ancre reste inchangée. Dès qu'au moins une articulation présente un déplacement supérieur ou égal à δ, la frame t est retenue et devient la nouvelle ancre ($t_{\text{ref}} \leftarrow t$). Ce filtrage séquentiel garantit que seules les frames contribuant à un mouvement significatif sont conservées : si le robot reste quasi-stationnaire pendant N frames consécutives, les $N - 1$ frames intermédiaires sont éliminées, tandis qu'un mouvement continu et rapide ne subit aucune suppression.

Pour déterminer la valeur du seuil δ, nous réalisons une analyse empirique des variations articulaires dans les données de téléopération. L'étude statistique des différences entre frames consécutives (en unités normalisées sur l'intervalle $[-100, 100]$ pour les articulations et $[0, 100]$ pour la pince) révèle des valeurs moyennes très faibles : 0.32 pour la rotation en lacet de l'épaule, 1.04 pour l'élévation de

l'épaule, 1.30 pour la flexion du coude, et 0.24 pour la rotation en roulis du poignet. Cette distribution s'explique par la prédominance des phases de maintien de position durant la téléopération. Cependant, les valeurs maximales atteignent 6.97 pour la flexion du coude et 5.66 pour l'élévation de l'épaule, correspondant aux phases de déplacement intentionnel. Après différents essais expérimentaux, nous avons fixé $\delta = 1.0$ en unités normalisées. Ce seuil permet d'éliminer suffisamment de frames quasi-statiques pour améliorer la fluidité lors de l'inférence, tout en préservant suffisamment de données d'entraînement représentant des mouvements significatifs pour l'apprentissage.

Nous proposons une étape de pré-traitement du dataset d'entraînement consistant à éliminer les frames redondantes avant l'apprentissage. L'analyse du dataset `sort-blocks` révèle que sur les 52 936 frames originales réparties en 43 épisodes, 7 171 frames (13.55%) sont identifiées comme redondantes selon notre critère. Après nettoyage, l'ensemble de données conserve 45 765 frames, soit une réduction moyenne de 166.8 frames redondantes par épisode.

Le modèle GR00T est ensuite entraîné sur ce dataset lissé avec les mêmes hyperparamètres que le modèle de référence (taille du lot 120, horizon de prédiction de 16, 20 000 steps), permettant une comparaison équitable des performances.

L'évaluation est réalisée sur 5 exécutions indépendantes de la tâche de tri pour chaque modèle. La figure 3.14 présente une comparaison détaillée des métriques de redondance pour chaque exécution.

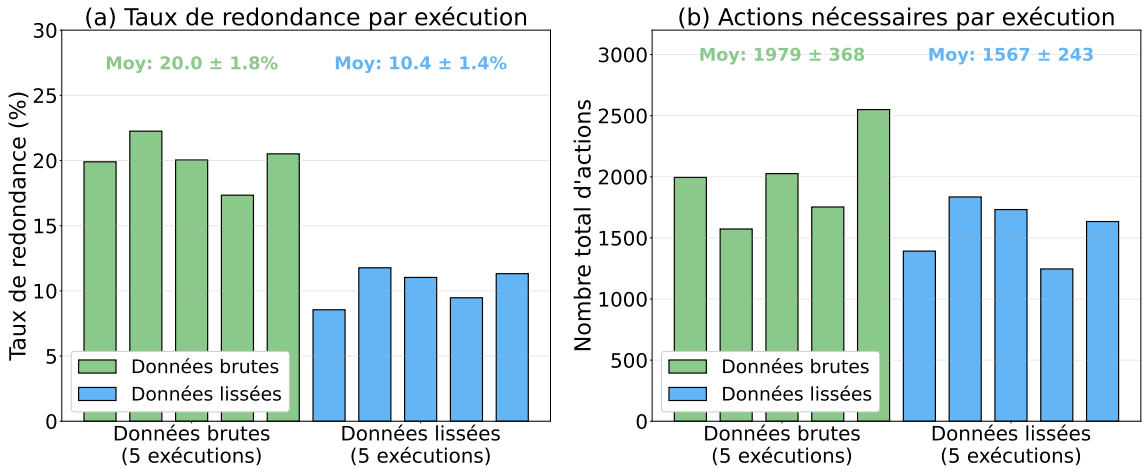


FIGURE 3.14 – Comparaison du taux de redondance et du nombre d'actions pour chaque exécution. Les lignes pointillées indiquent les moyennes respectives. Le modèle entraîné sur données lissées présente systématiquement de meilleures performances sur l'ensemble des exécutions.

Le modèle entraîné sur données brutes présente un taux de redondance moyen de 20.0% ($\sigma = 1.8$), avec des valeurs individuelles variant de 17.3% à 22.3%. En comparaison, le modèle entraîné sur données lissées affiche un taux moyen de 10.4% ($\sigma = 1.4$), avec des valeurs variant de 8.6% à 11.8%. Cette réduction de 47% du taux de redondance témoigne de l'efficacité du pré-traitement.

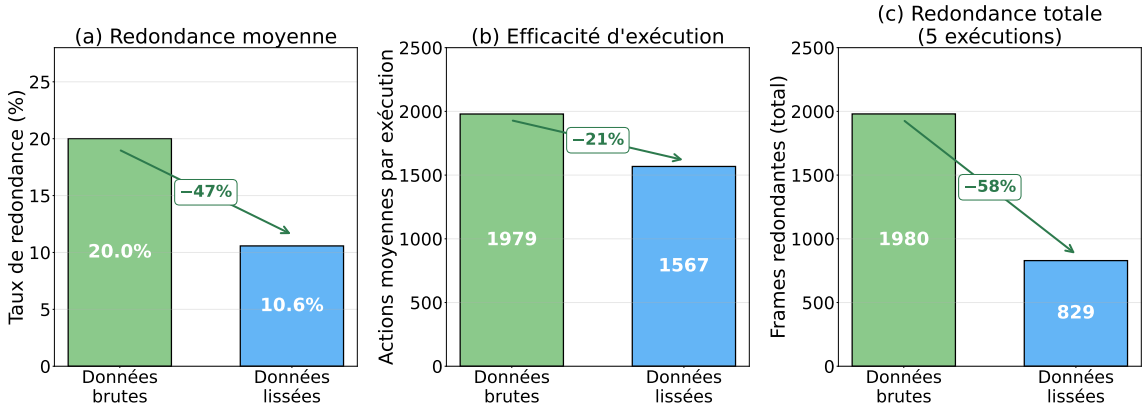


FIGURE 3.15 – Synthèse des améliorations apportées par le pré-traitement des données. (a) Réduction de 47% du taux de redondance moyen. (b) Réduction de 21% du nombre d'actions nécessaires. (c) Réduction de 58% des frames redondantes totales sur 5 exécutions.

La figure 3.15 synthétise les résultats agrégés. Sur l'ensemble des 5 exécutions, le modèle original génère 1 980 frames redondantes sur 9 897 frames totales, tandis que le modèle lissé n'en produit que 829 sur 7 839 frames, soit une réduction absolue de 58% des mouvements parasites.

Au-delà de la réduction de la redondance, le nombre total d'actions nécessaires pour accomplir la tâche diminue également : le modèle lissé requiert en moyenne 1 568 actions par exécution contre 1 979 pour le modèle original, soit une réduction de 21%. Cette efficacité accrue se traduit par des trajectoires plus directes et un temps d'exécution réduit, sans dégradation du taux de succès qui reste comparable entre les deux configurations.

Ces résultats démontrent que la qualité des données d'entraînement influence directement la fluidité des mouvements générés. Le pré-traitement par élimination des frames redondantes constitue une technique simple mais efficace pour améliorer le comportement dynamique des politiques VLA, produisant des trajectoires plus fluides et plus efficientes sans nécessiter de modifications architecturales.

3.7 Limitations et positionnement par rapport aux méthodes déterministes

Malgré les performances prometteuses observées lors de nos expérimentations, l'intégration des modèles VLA pour le contrôle robotique présente plusieurs limitations qu'il convient d'analyser. Ces contraintes, d'ordre matériel, méthodologique et applicatif, définissent les frontières actuelles de cette technologie. Afin de situer objectivement ces limitations, nous les comparons systématiquement aux méthodes déterministes conventionnelles reposant sur la détection de pose, la cinématique inverse et la planification de trajectoire. Cette mise en perspective permet d'évaluer si le gain de flexibilité offert par les VLA justifie les compromis en termes de précision, de latence et de coût matériel.

3.7.1 Dépendance à la calibration physique

Contrairement à l'intuition suggérant qu'un modèle prenant en entrée des images devrait généraliser à différentes configurations physiques, les modèles VLA produisent des commandes articulaires absolues et non relatives, et sont ainsi intrinsèquement liés à la calibration spécifique du robot sur lequel ils ont été entraînés. Le modèle ne maintient pas de représentation interne de la géométrie du robot ni de sa position dans l'espace suite à ses mouvements précédents, mais apprend une correspondance directe entre les observations visuelles, les instructions textuelles et les positions articulaires cibles. Ainsi, pour les robots assemblés manuellement où les calibrations diffèrent inévitablement, même pour des unités du même modèle, une phase de collecte de données suivie d'un ajustement fin (*fine-tuning*) est requise pour chaque nouvelle configuration.

3.7.2 Limitations de l'apprentissage multi-tâches

En théorie, les modèles VLA devraient généraliser à plusieurs tâches à partir d'un unique point de contrôle (*checkpoint*), en adaptant leur comportement selon les instructions textuelles. L'objectif serait de disposer d'un modèle unique capable d'interpréter diverses consignes et d'adapter ses actions en conséquence, sans réentraînement. Cependant, cette capacité multi-tâches n'est pas encore implémentée dans les architectures open source actuelles. En pratique, le déploiement sur une nouvelle tâche nécessite toujours un ajustement fin spécifique, ce qui limite l'avantage d'un modèle véritablement généraliste et impose un cycle complet de collecte de données et d'ajustement pour chaque application.

3.7.3 Adaptation aux variations environnementales

Bien que les VLA soient contraints par leur configuration matérielle, ils généralisent aux variations de l'environnement là où les systèmes déterministes échouent. Héritant de la compréhension sémantique de leurs modèles de fondation vision-langage, les VLA s'adaptent aux changements d'éclairage, aux nouvelles orientations d'objets et aux instructions reformulées en langage naturel. L'intégration d'une nouvelle tâche ne requiert que la collecte de démonstrations supplémentaires suivies d'un ajustement fin, ces nouvelles trajectoires renforcent par ailleurs la robustesse du modèle sur les conditions déjà maîtrisées.

À l'inverse, les systèmes déterministes sont sensibles à toute déviation par rapport aux conditions de calibration : un objet déplacé ou une variation d'éclairage suffisent à invalider les seuils de détection calibrés, provoquant l'échec de l'opération. De plus, l'intégration d'un nouvel objet ou d'une nouvelle tâche requiert une reprogrammation explicite par des experts, chaque nouvelle capacité nécessitant une recalibration du pipeline de perception et de contrôle.

3.7.4 Considérations de sécurité

La nature probabiliste des modèles VLA soulève des préoccupations en matière de sécurité opérationnelle. Contrairement aux systèmes de contrôle déterministes où une trajectoire défaillante peut être identifiée et corrigée de manière reproductible, les politiques VLA déployées hors de leur distribution d'entraînement peuvent générer des mouvements erratiques et imprévisibles. Ce comportement stochastique nécessite la mise en place de mécanismes de supervision, limites articulaires logicielles, détection de collisions, arrêts d'urgence, supervision humaine, pour prévenir les dommages matériels ou les risques pour les opérateurs à proximité.

3.7.5 Contraintes matérielles et énergétiques

L'exécution des modèles VLA requiert des ressources computationnelles substantielles. Bien que les latences d'inférence mesurées soient compatibles avec le contrôle en temps réel (30 Hz), le déploiement de ces modèles nécessite un GPU moderne disposant d'au minimum 6 Go de VRAM pour les architectures les plus légères comme SmolVLA, et jusqu'à 12 Go pour les modèles de grande taille comme GR00T ou Pi0.5. L'entraînement et l'ajustement fin requiert des configurations multi-GPU disposant de 80 Go ou plus de VRAM par carte, avec des durées

de calcul de plusieurs jours. La consommation énergétique d'un GPU d'inférence dépasse les 300 W pour une NVIDIA RTX 4090 utilisée pour nos évaluations.

Cette exigence matérielle contraste fortement avec les systèmes déterministes classiques, qui s'exécutent sur des automates programmables industriels (PLC) ou des contrôleurs temps réel embarqués, équipés de processeurs ARM ou x86 dédiés, sans GPU. Ces contrôleurs consomment typiquement entre 5 et 20 W et coûtent moins chers que les GPU modernes. La vision industrielle associée utilise des modèles légers exécutables sur CPU ou sur des accélérateurs embarqués à faible consommation.

3.7.6 Latence et cadence industrielle

La cadence est déterminante pour l'adoption industrielle, où le temps de cycle conditionne directement la productivité de la ligne de production. Les boucles de contrôle des PLC s'exécutent à des fréquences de 500 à 2 000 Hz, avec des latences inférieures à la milliseconde. Les contrôleurs de robots industriels assurent un contrôle servo à 250-1 000 Hz, garanti par des systèmes d'exploitation temps réel certifiés (RTOS). La planification de trajectoire ajoute une latence de 10 à 100 ms, mais cette opération n'est effectuée qu'une seule fois avant l'exécution, permettant des temps de cycle souvent inférieurs à 10 secondes pour un pick-and-place standard.

En comparaison, comme démontré dans nos expériences, les modèles VLA fonctionnent à des fréquences de contrôle de l'ordre de 10 à 50 Hz après optimisation. La latence d'inférence d'un passage complet (noyau vision-langage et tête d'action) se situe typiquement entre 30 et 150 ms selon l'architecture et le matériel. La segmentation d'actions réduit la fréquence des appels au modèle (environ 2 Hz dans notre configuration avec $C = 16$). Ces fréquences restent compatibles avec des contrôleurs temps réel, mais sont insuffisantes pour le contrôle dynamique à haute fréquence ou pour les applications nécessitant des temps de cycle très courts.

3.7.7 Précision et répétabilité

Les robots industriels standard atteignent une répétabilité de ±0,02 mm à ±0,4 mm grâce à des encodeurs de haute résolution, des modèles cinématiques calibrés et une planification de trajectoire déterministe. Ce comportement est strictement identique sur des millions de cycles et conforme à la norme ISO 9283. Cette précision sub-millimétrique est indispensable pour des tâches telles que l'assemblage de composants électroniques, le soudage de précision ou l'usinage.

Les modèles VLA, en revanche, génèrent des actions par des processus probabilistes qui introduisent une variance inhérente : deux exécutions identiques produisent des trajectoires différentes. Cette variabilité limite les VLA à des tâches ayant des tolérances de positionnement de l'ordre du millimètre ou supérieures. Les résultats de nos expérimentations illustrent cette limitation : les échecs se concentrent dans les configurations nécessitant une précision millimétrique, notamment la préhension d'objets de petite taille aux limites de l'espace de travail. Les VLA conviennent donc aux opérations de manipulation où la tolérance positionnelle est de l'ordre du centimètre, mais ne peuvent pas se substituer à un contrôle cinématique pour les tâches de haute précision.

En conclusion, les modèles VLA et les méthodes déterministes répondent à des besoins complémentaires. Les systèmes déterministes restent indispensables pour les tâches répétitives à haute cadence exigeant une précision sub-millimétrique et une certification de sécurité formelle. Les VLA, en revanche, ouvrent l'automatisation à des tâches auparavant inaccessibles à la robotique traditionnelle : environnements non structurés, forte variabilité des objets, co-manipulation avec supervision humaine, et applications où le coût de reprogrammation dépasse celui de l'adaptation par apprentissage.

3.8 Conclusion

Ce chapitre a présenté l'ensemble du processus d'intégration d'un VLA pour le contrôle autonome du robot SO-101. Nous avons commencé par détailler la procédure de calibration des bras robotiques, établissant le pipeline de transformation bidirectionnelle entre les valeurs brutes des encodeurs et les représentations normalisées utilisées par les modèles d'apprentissage profond.

La collecte de données par téléopération a permis de constituer trois jeux de données distincts au format LeRobotDataset v3.0 : `pick-place-red-block` (239 épisodes), `sort-blocks` (43 épisodes) et `arrange-table` (73 épisodes), couvrant des tâches de manipulation de complexité croissante. Nous avons ensuite présenté la configuration d'entraînement, incluant les stratégies de normalisation, la segmentation d'actions, les fonctions de perte et les hyperparamètres d'optimisation.

L'évaluation comparative des architectures GR00T, SmolVLA et Pi0.5 sur un protocole standardisé de 100 configurations de test a révélé la supériorité de GR00T avec un taux de succès de 93%, contre 62% pour Pi0.5 et 61% pour SmolVLA. L'augmentation de la taille de lot de 64 à 120 a permis d'atteindre 97% pour GR00T

validant, dans le contexte des modèles VLA, que des lots plus grands favorisent une meilleure généralisation. L'étude de l'impact de l'horizon de prédiction (*chunk size*) a démontré qu'un horizon court (16 actions) favorise la réactivité et améliore les performances par rapport à un horizon étendu (50 actions). Par ailleurs, pour permettre l'inférence sur des machines plus restreintes en mémoire graphique, nous avons appliqué la distillation de connaissances à SmolVLA (428M paramètres) à partir de GR00T (3B paramètres), ce qui a conduit à une amélioration notable du taux de succès, passant de 61% à 83%. Enfin, le pré-traitement des données par élimination des frames redondantes a réduit de 47% le taux de micro-tremblements lors de l'inférence.

Pour les modèles finaux déployés sur les trois jeux de données, nous avons retenu l'architecture GR00T entraînée avec une taille de lot de 120, un horizon de prédiction de 16 et 10 000 étapes d'optimisation.

Le système étant désormais doté de capacités de manipulation autonome via le modèle VLA, le chapitre 4 présentera le développement du module de perception visuelle pour l'inspection industrielle, en proposant l'approche SSL-YOLO pour la détection de défauts de surface.

Chapitre 4

Systèmes de détection de défauts pour l'inspection industrielle

4.1 Introduction

Comme présenté au chapitre 1, la détection de défauts de surface constitue un élément clé de notre système robotique intelligent. Les résultats fournis par le module de vision peuvent déclencher des comportements spécifiques du robot : changement de trajectoire, tri de pièces, ajustement de la stratégie de *pick-and-place*, ou notification à un opérateur humain.

Dans ce chapitre, nous étudions des approches visant à améliorer les performances des systèmes de détection de défauts à la fois sur des classes bien représentées et sur des classes rares, en nous appuyant sur le concept d'apprentissage *few-shot* (Sun et coll. 2022) et sur l'apprentissage contrastif auto-supervisé.

Ce chapitre est structuré en deux parties complémentaires :

— Dans une première partie, nous réalisons une comparaison expérimentale de plusieurs modèles modernes de détection d'objets en temps réel (famille YOLO et RT-DETR) appliqués à la détection de défauts de surface sur le jeu de données NEU-DET. Cette étude permet d'identifier les compromis entre précision, vitesse d'inférence et complexité de déploiement dans un contexte industriel temps réel.

— Dans une seconde partie, nous proposons une solution basée sur l'apprentissage semi-supervisé et l'apprentissage contrastif pour améliorer les performances dans un contexte *few-shot*, en particulier sur des classes de défauts rares.

L'ensemble de ces contributions s'inscrit dans la perspective d'une intégration fluide du module de détection de défauts dans l'architecture globale du système robotique, où un agent (basé sur un LLM) peut invoquer dynamiquement ce module sur une image de la scène, interpréter les résultats (présence, classe et localisation de défauts) et adapter le comportement du robot en conséquence.

4.2 Comparaison des modèles de détection d'objets et de défauts de surface

Dans cette section, nous évaluons six modèles de détection d'objets en temps réel présentés au chapitre 1 : YOLOv5 (Ultralytics 2021), YOLOv6 v3.0 (Li et coll. 2023a), YOLOv7 (Wang et coll. 2023a), YOLOv8 (Varghese et coll. 2024), YOLOv9 (Wang et coll. 2024b), et RT-DETR (Lv et coll. 2023). L'objectif est d'identifier les compromis entre précision, vitesse d'inférence et complexité de déploiement dans notre contexte industriel temps réel.

Le tableau 4.1 résume les caractéristiques architecturales principales des modèles évalués :

Tableau 4.1 – Vue d'ensemble des variantes de modèles étudiés à grande échelle. #Params (M) : nombre de paramètres (en millions). FLOPs (G) : opérations en virgule flottante (en milliards). Noyau (*Backbone*) : réseau d'extraction de caractéristiques. Module de fusion (*Neck*) : composant de fusion multi-échelle.

Modèle	#Params (M)	FLOPs (G)	Noyau	Module de fusion	Paradigme de détection (tête)
YOLOv5 (Ultralytics 2021)	46.5	109.1	CSP-Darknet53	SPPF et PANet	Basé sur ancres
YOLOv6 v3.0 (Li et coll. 2023a)	59.6	150.7	EfficientRep	RepBi-PAN	Entraînement assisté par ancres (AAT)
YOLOv7 (Wang et coll. 2023a)	36.9	104.7	E-ELAN	FPN	Basé sur ancres
YOLOv8 (Varghese et coll. 2024)	43.7	165.2	CSP-Darknet53	SPPF et PANet avec module C2f	Sans ancre
YOLOv9 (Wang et coll. 2024b)	58.1	192.5	GELAN	FPN-ICN	Sans ancre
RT-DETR (Lv et coll. 2023)	32	110	HGNetv2	Encodeur-Décodeur hybride efficace	Sans ancre

4.2.1 Jeu de données NEU-DET

Le jeu de données NEU-DET (Song et coll. 2013), développé à la Northeastern University (NEU), est une référence pour la détection de défauts de surface sur des bandes d'acier laminées à chaud. Il comprend 1800 images en niveaux de gris, chacune de résolution 200×200 pixels, organisées en six catégories de défauts typiques : incrustation de calamine (*Rolled-in Scale*, Rs), tâches (*Patches*, Pa), craquelures (*Crazing*, Cr), surface piquée (*Pitted Surface*, Ps), inclusion (*Inclusion*,

In) et rayures (*Scratches*, Sc). Chaque catégorie contient 300 images distinctes, généralement réparties en 240 images pour l'entraînement et 60 pour le test.

NEU-DET est largement utilisé comme banc d'essai pour l'évaluation des systèmes de détection de défauts industriels. La revue systématique de Ma et coll. (Ma et coll. 2024, Fig. 1) rapporte qu'il est utilisé dans 43,4% des travaux examinés, devant GC10-DET (17%) et MVTec AD (9,4%). Cette popularité en fait un support pertinent pour comparer les modèles de détection d'objets en temps réel dans notre contexte.

Les six classes de défauts sont décrites dans le tableau 4.2.

Tableau 4.2 – Classes de défauts de surface dans l'acier

Défaut	Nomenclature en français	Description technique
Crazing	Craquelures	Défauts de surface résultant de tensions dans le matériau.
Inclusion	Inclusion	Présence de matériaux étrangers à l'intérieur de la matrice d'acier.
Patches	Tâches	Zones décolorées ou irrégulières sur la surface.
Pitted Surface	Surface piquée	Surface présentant des creux ou des trous dus à la corrosion ou à l'usure.
Rolled-in Scale	Incrustation de calamine	Formation d'oxyde de fer durant la production, restant incrustée à la surface.
Scratches	Rayures	Marques linéaires dues à des frottements ou des impacts sur la surface.

Des exemples visuels de ces différents défauts sont présentés dans la figure 4.1.

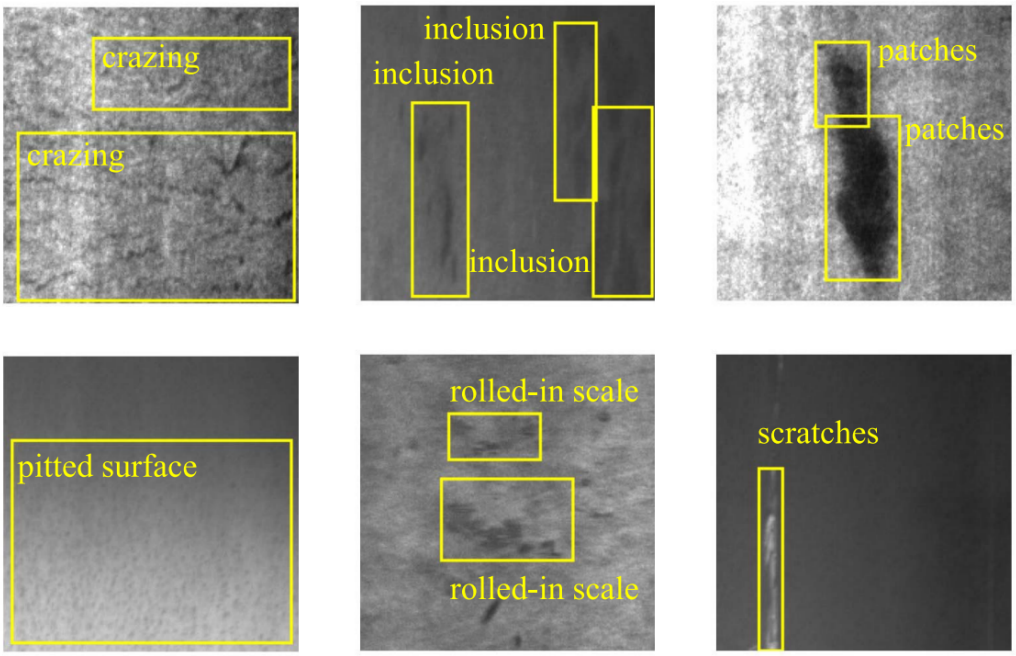
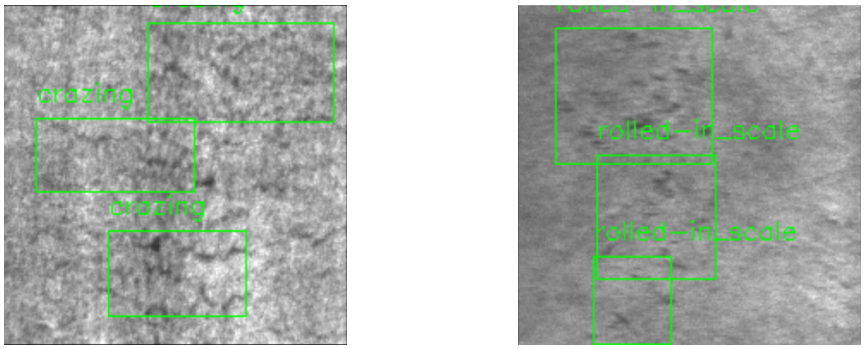


FIGURE 4.1 – Exemples de défauts de surface en acier.

À partir de ces exemples, on observe la diversité des textures, des formes et des apparences à l'intérieur d'une même classe de défauts. La tâche de détection est complexifiée par le faible contraste des images en niveaux de gris, en particulier pour les craquelures et les incrustations de calamine, ainsi que par la résolution limitée (200×200 pixels), comme illustré dans la figure 4.2.



(a) Exemple de craquelures (Cr).

(b) Exemple d'incrustation de calamine (Rs).

FIGURE 4.2 – Exemples de défauts de surface difficiles à identifier dans NEU-DET.

Ces caractéristiques font de NEU-DET un jeu de données exigeant mais pertinent pour évaluer la capacité des modèles à généraliser dans des conditions proches de l'industrie.

4.2.2 Métriques d'évaluation des modèles de détection

Les métriques d'évaluation utilisées pour comparer les systèmes de détection de défauts de surface sont les suivantes :

a) Intersection sur l'Union (IoU)

L'Intersection over Union (IoU) mesure le degré de superposition entre la vraie boîte englobante et la boîte prédite :

$$\text{IoU} = \frac{\text{Surface d'intersection}}{\text{Surface d'union}} = \frac{S(b_i \cap b_{i,\text{pred}})}{S(b_i \cup b_{i,\text{pred}})} \tag{4.1}$$

où b_i est la boîte réelle et $b_{i,\text{pred}}$ la boîte prédite correspondante.

b) Précision moyenne (AP)

La précision moyenne AP est définie comme l'aire sous la courbe précision-rappel, pour un seuil fixé d'IoU (par exemple 0,5) :

$$AP(\text{IoU}_{\text{thresh}}) = \int_0^1 p(r, \text{IoU}_{\text{thresh}})\, dr \tag{4.2}$$

où $p(r, \text{IoU}_{\text{thresh}})$ est la précision à un rappel r, en ne considérant que les paires de boîtes (vraies et prédites) dont l'IoU est supérieur ou égal à $\text{IoU}_{\text{thresh}}$.

On note parfois AP_S, AP_M et AP_L pour la précision moyenne sur les petits, moyens et grands objets.

c) Moyenne de la précision moyenne (mAP)

La mAP (Mean Average Precision) est la moyenne de AP sur l'ensemble des classes :

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{4.3}$$

où N est le nombre de classes et AP_i la précision moyenne pour la classe i. La notation mAP@50 désigne la mAP calculée avec un seuil d'IoU de 0,5.

d) Précision, rappel et F1-score

En utilisant les taux de vrais positifs (TP), faux positifs (FP), vrais négatifs (TN) et faux négatifs (FN), on définit :

$$R = \frac{TP}{TP + FN} \tag{4.4}$$

$$P = \frac{TP}{TP + FP} \tag{4.5}$$

$$F1 = 2 \cdot \frac{P \times R}{P + R}. \tag{4.6}$$

e) Temps d'inférence et FPS

Le temps d'inférence (en millisecondes) correspond à la durée nécessaire à un modèle entraîné pour prédire les boîtes et classes à partir d'une image. Il dépend notamment de la taille du modèle, de la résolution d'entrée et du matériel utilisé. Le temps d'inférence permet de calculer le nombre d'images traitées par seconde (FPS, Frames Per Second).
Un FPS élevé est indispensable pour l'inspection en ligne couplée au bras robotique.

4.2.3 Analyse comparative des performances

Pour chaque modèle, nous menons deux expériences :

— Images redimensionnées à 320×320 pixels, taille de lot 64.

— Images redimensionnées à 640×640 pixels, taille de lot 32.

Le nombre d'époques est fixé à 100 dans tous les cas. Les augmentations de données par défaut de chaque implémentation sont utilisées. Toutes les évaluations ont été réalisées sur GPU NVIDIA GeForce RTX 4090.

Sur NEU-DET, l'analyse des résultats présentée à la figure 4.3 montre que :

— YOLOv6 v3.0 obtient la meilleure mAP@50 avec 0,772 sur les deux résolutions (320 px et 640 px), tout en conservant une vitesse d'inférence élevée (440 FPS à 320 px et 160 FPS à 640 px).

— YOLOv9-c suit de près avec une mAP@50 de 0,755 à 320 px (255 FPS) et 0,746 à 640 px (90 FPS), offrant une bonne précision mais le FPS le plus faible parmi les modèles YOLO.

— YOLOv7 se distingue par un excellent FPS (500 FPS à 320 px), ce qui en fait un bon candidat pour des scénarios très contraints en temps réel, avec une mAP@50 de 0,744 à 320 px.

— YOLOv8 L atteint une mAP@50 de 0,726 à 320 px (360 FPS) et 0,734 à 640 px (165 FPS), soit environ 5,2% de moins que YOLOv6 v3.0 à résolution 320 px.

— RT-DETR, bien qu'exempt de suppression non maximale (NMS), reste relativement coûteux en calcul (Transformers de bout en bout) et présente un FPS plus faible avec une mAP@50 de seulement 0,698 à 320 px.

L'impact de la résolution sur la mAP@50 varie selon l'architecture. YOLOv6 v3.0 maintient une performance stable (0,772) indépendamment de la résolution. YOLOv8 bénéficie légèrement de la résolution supérieure (+1,1% de 320 à 640 px). En revanche, YOLOv7, YOLOv9 et RT-DETR subissent une baisse de mAP@50 lorsque la résolution augmente (−8,1%, −1,2% et −1,7% respectivement), ce qui suggère que le sur-échantillonnage d'images de basse résolution n'apporte pas systématiquement un bénéfice.

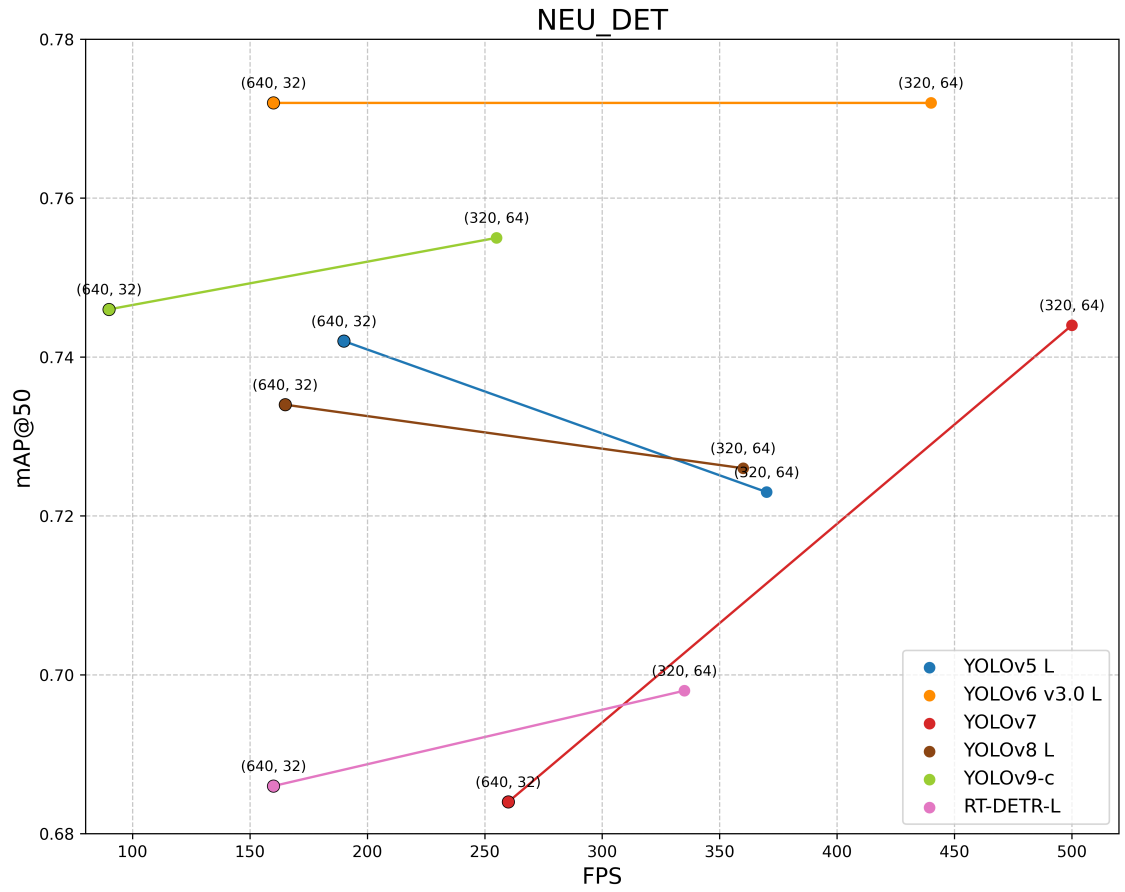


FIGURE 4.3 – Performances des modèles à grande échelle sur NEU-DET en termes de FPS et mAP@50.

L'analyse par classe, illustrée à la figure 4.4, révèle que YOLOv6 v3.0 tend à mieux gérer les défauts difficiles (Rs, Cr, In) sur images 640 × 640, tandis que YOLOv7 et YOLOv9 sont plus performants sur 320 × 320.

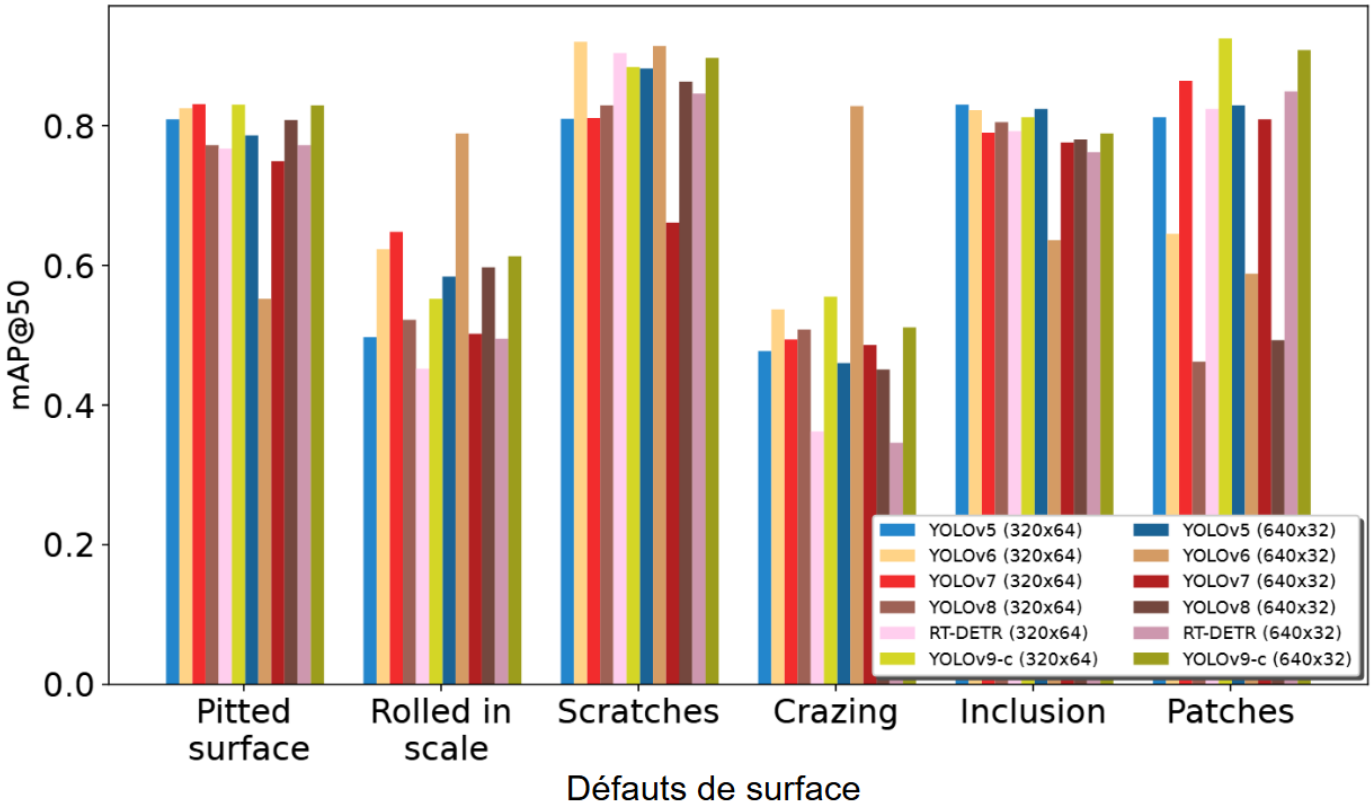


FIGURE 4.4 – Analyse par classe de la mAP@50 sur NEU-DET pour des résolutions 320 px et 640 px.

En résumé, l'étude comparative met en évidence que :

— Tous les modèles atteignent des vitesses d'inférence compatibles avec une intégration temps réel dans un pipeline industriel.

— YOLOv6 v3.0 offre le meilleur compromis précision/vitesse sur NEU-DET.

— La famille YOLO reste globalement plus adaptée que RT-DETR pour un déploiement temps réel.

4.3 Solution basée sur l'apprentissage auto-supervisé contrastif (SSL-YOLO)

L'apprentissage *few-shot* permet de reconnaître de nouveaux objets ou défauts après seulement quelques exemples, alors que les modèles d'apprentissage profond classiques nécessitent de grandes quantités de données annotées. Dans le contexte

industriel, la collecte et l'annotation de nouvelles catégories de défauts sont coûteuses et difficiles à systématiser avec certaines classes qui restent naturellement rares.

Les approches *few-shot* et semi-supervisées exploitent un grand volume de données non étiquetées, complété par un petit nombre d'échantillons annotés par classe, afin de construire des représentations plus robustes et généralisables. Ce paradigme est particulièrement pertinent pour le jeu de données NEU-DET et, plus largement, pour des scénarios où l'on souhaite rapidement adapter un système de détection à une nouvelle classe de matériau ou de défaut sans devoir ré-annoter un corpus complet.

Dans notre architecture globale, être capable d'adapter ce module à de nouvelles classes de défauts ou à de nouvelles pièces avec peu d'exemples annotés est un atout majeur pour envisager un déploiement industriel flexible et évolutif.

Pour cela, nous proposons une approche en deux étapes :

1. **Pré-entraînement auto-supervisé contrastif** sur des données non annotées, afin d'apprendre un noyau (*backbone*) robuste aux variations de texture, de contraste et de conditions d'éclairage.
2. **Ajustement fin (*fine-tuning*) supervisé *few-shot*** sur un jeu réduit d'images annotées (10-shot par classe nouvelle), en gelant le noyau et en ajustant seulement la tête de détection.

Nous appelons la solution résultante SSL-YOLO, en référence à l'apprentissage auto-supervisé (Self-Supervised Learning, SSL) utilisé lors du pré-entraînement du noyau.

L'apprentissage contrastif, qui constitue le cœur de notre pré-entraînement, cherche à apprendre des représentations (embeddings) dans lesquelles les échantillons d'une même instance sont proches tandis que ceux d'instances différentes sont éloignés. Dans notre cas, pour la phase auto-supervisée, nous ne disposons pas d'étiquettes de classe. Les "étiquettes" sont donc implicites : deux vues augmentées de la même image (paire positive) doivent être proches, tandis que les vues issues d'images différentes (paires négatives) doivent être éloignées. Cela permet d'apprendre, sans annotation, des représentations sensibles aux structures locales de la surface (textures, micro-défauts, variations de contraste).

4.3.1 Choix de YOLOv8 comme base pour SSL-YOLO

Bien que YOLOv6 v3.0 atteigne la meilleure mAP@50 sur NEU-DET, nous avons choisi YOLOv8 comme base pour SSL-YOLO pour des raisons pratiques liées au développement et au déploiement :

— **Accessibilité de la documentation** Bien que le dépôt YOLOv6 v3.0 (Meituan) ait progressé vers une documentation anglophone, certaines ressources techniques restent partiellement en chinois. YOLOv8 (Ultralytics) bénéficie d'une documentation exhaustive, d'une maintenance active et d'une large communauté facilitant le débogage et les adaptations.

— **Implémentation PyTorch modulaire** L'architecture YOLOv8 est implémentée de manière modulaire, rendant plus accessibles les modifications nécessaires pour le développement de notre solution.

La figure 4.5 illustre l'architecture modulaire de YOLOv8, composée de trois blocs principaux : un noyau convolutif pour l'extraction de caractéristiques à différentes échelles, un module de fusion (*neck*) pour l'agrégation multi-échelle de ces caractéristiques, et une tête de détection sans ancres (*anchor-free*) pour les prédictions de boîtes englobantes et de classes.

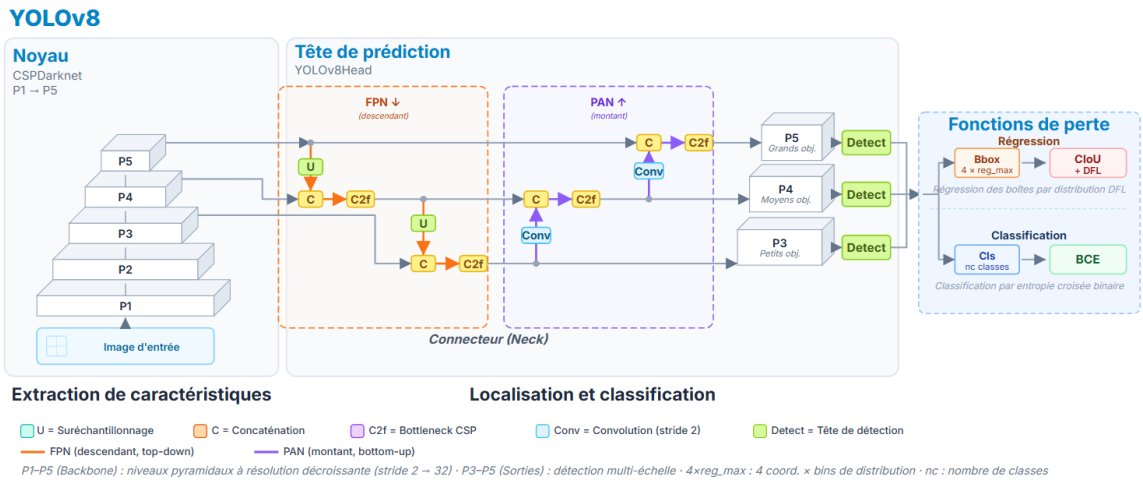


FIGURE 4.5 – Architecture simplifiée de YOLOv8 (VARGHESE et coll. 2024).

4.3.2 Pré-entraînement contrastif du noyau

Le pré-entraînement contrastif constitue l'étape centrale de notre approche SSL-YOLO. Il s'appuie sur un framework inspiré de SimCLR (CHEN et coll. 2020), illustré à la figure 4.6, pour apprendre des représentations visuelles riches à partir d'images non annotées.

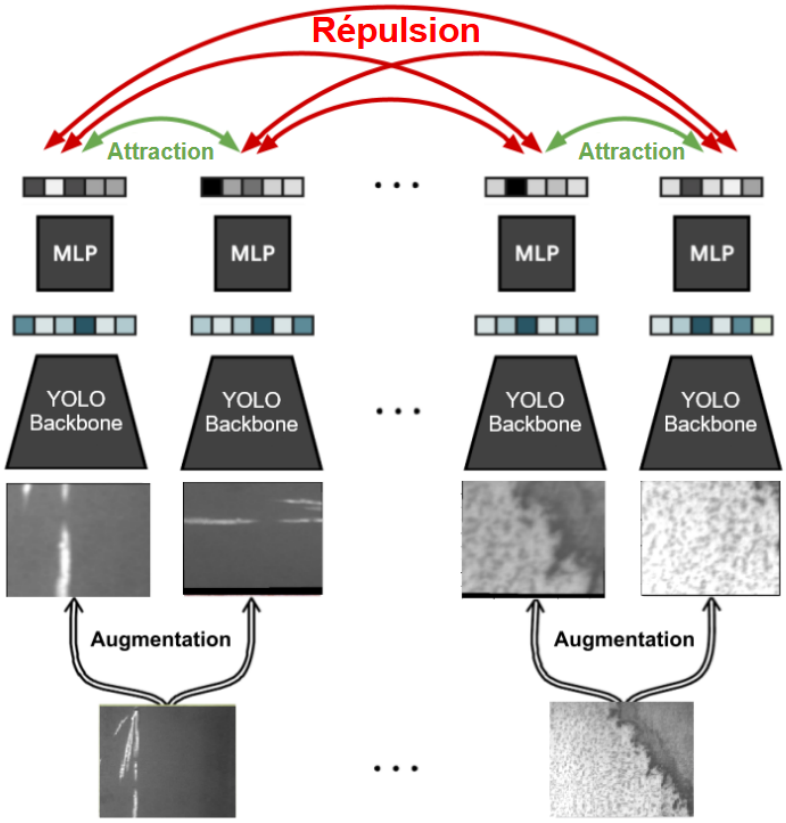


FIGURE 4.6 – Schéma du framework d'apprentissage contrastif SimCLR.

À chaque itération du pré-entraînement :

1. Un lot d'images non annotées des défauts de base abondants est échantillonné.
2. Pour chaque image, deux vues augmentées distinctes sont générées à l'aide du pipeline de transformations détaillé dans le tableau 4.3.
   — Les deux vues d'une même image forment une **paire positive** (ancre-positif).
   — Les vues issues d'images différentes constituent des **paires négatives**.
3. Le noyau extrait un vecteur de caractéristiques de haut niveau pour chacune des vues.
4. Un réseau MLP de projection (Section c)) transforme ces vecteurs en représentations de dimension réduite.
5. Une fonction de perte contrastive NT-Xent (Section d)) est utilisée pour rapprocher les représentations des paires positives et éloigner celles des paires négatives.

L'objectif est d'apprendre un noyau sensible aux variations de textures et de contrastes caractéristiques des défauts de surface, tout en restant robuste aux changements d'éclairage, de point de vue ou à certains bruits.

a) Augmentations contrastives

Le pipeline d'augmentation génère, pour chaque image ancre, deux vues stochastiquement différentes. Le tableau 4.3 détaille les transformations appliquées séquentiellement :

TABLEAU 4.3 – Pipeline d'augmentations utilisé pour le pré-entraînement contrastif.

Transformation	**Paramètres**	**Description**
`RandomResizedCrop`	échelle : [0,7, 1,0]	Recadrage aléatoire suivi d'un redimensionnement à 224 × 224 pixels.
`CLAHE`	—	Égalisation adaptative d'histogramme pour renforcer le contraste local.
`ColorJitter`	luminosité : 0,1, contraste : 0,2	Perturbations aléatoires de luminosité et de contraste.
`RandomRotation`	±30	Rotation aléatoire jusqu'à 30.
`RandomHorizontalFlip`	probabilité = 0,5	Retournement horizontal aléatoire.
`RandomVerticalFlip`	probabilité = 0,3	Retournement vertical aléatoire.
`RandomAffine`	translation : (0,05, 0,05)	Translation aléatoire de ±5% de la taille de l'image.
`GaussianBlur`	noyau : $[3, 5]$, σ : [0,1, 2,0]	Flou gaussien aléatoire pour simuler des variations de mise au point.
`RandomPerspective`	distorsion : 0,2, probabilité = 0,3	Transformation perspective aléatoire simulant un changement d'angle de vue.
`RandomGrayscale`	probabilité = 0,2	Conversion aléatoire en niveaux de gris, forçant le modèle à exploiter les structures spatiales.

La figure 4.7 illustre un exemple concret de paire positive obtenue par ce pipeline d'augmentation.

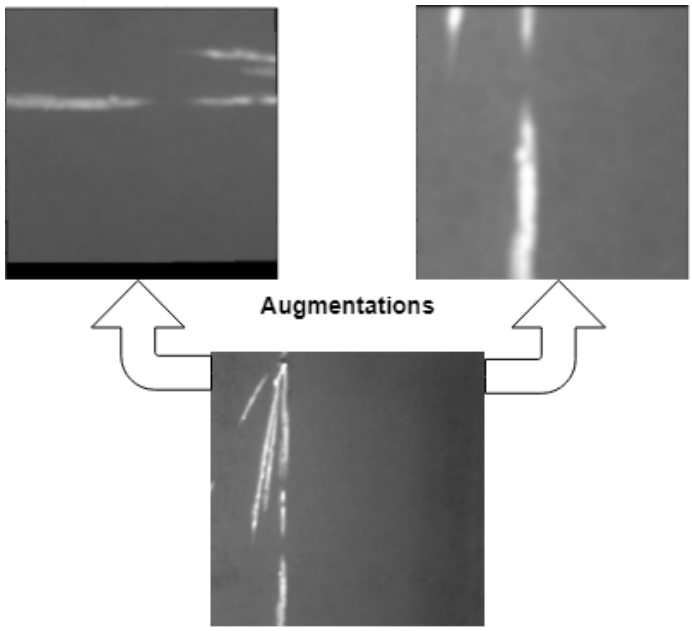


FIGURE 4.7 – Exemple de paire positive augmentée à partir d'une image ancre.

b) Hyperparamètres du pré-entraînement contrastif

Le tableau 4.4 résume les hyperparamètres utilisés lors du pré-entraînement contrastif :

TABLEAU 4.4 – Hyperparamètres du pré-entraînement contrastif de SSL-YOLO.

Hyperparamètre	**Valeur**	**Description**
Modèle de base	YOLOv8-L	Variante *Large* de YOLOv8 utilisée comme noyau.
Couches du noyau	11	Nombre de couches du noyau entraînées en auto-supervision.
Taille d'image	224×224	Résolution d'entrée après redimensionnement.
Taille de lot	120	Nombre d'images par lot. Un lot plus grand fournit davantage de paires négatives.
Nombre d'époques	200	Nombre maximal d'époques d'entraînement.
Taux d'apprentissage	5×10^{-4}	Taux d'apprentissage initial de l'optimiseur AdamW.
Optimiseur	AdamW	Optimiseur avec régularisation (*weight decay*) intégrée.
Patience (arrêt précoce)	20 époques	Nombre d'époques sans amélioration avant l'arrêt de l'entraînement.

c) Architecture de la tête MLP de projection

La tête MLP sert à projeter les cartes de caractéristiques de sortie du noyau dans un espace de dimension réduite adapté à la perte contrastive. L'architecture utilisée est un MLP à deux couches avec normalisation par lot :

```
self.mlp = nn.Sequential(
    nn.AdaptiveAvgPool2d((1, 1)),
    nn.Flatten(),
    nn.Linear(in_features, hidden_dim, bias=False),
    nn.BatchNorm1d(hidden_dim),
    nn.GELU(),
    nn.Linear(hidden_dim, out_features, bias=True),
    nn.BatchNorm1d(out_features, affine=False),
)
```

Pour YOLOv8-L, les dimensions sont : `in_features` = 512, `hidden_dim` = 512, `out_features` = 256. Le rôle de chaque couche est le suivant :

- `AdaptiveAvgPool2d((1, 1))` : Effectue un pooling moyen adaptatif sur la carte de caractéristiques, produisant un vecteur global par image.
- `Flatten()` : Aplatit le tenseur en un vecteur 1D.
- `Linear(512, 512)` + `BatchNorm1d` + `GELU()` : Première couche de projection avec normalisation par lot et activation GELU, introduisant de la non-linéarité.
- `Linear(512, 256)` + `BatchNorm1d` : Seconde couche réduisant la dimension à 256 avec une normalisation finale sans paramètres apprenables (`affine=False`), ce qui facilite la comparaison par similarité cosinus et limite le coût de calcul de la perte NT-Xent.

d) Fonction de perte contrastive NT-Xent

Nous utilisons la NT-Xent Loss (Normalized Temperature-scaled Cross Entropy Loss), également connue sous le nom d'InfoNCE, largement utilisée dans les travaux SimCLR et MoCo (Chen et coll. 2020).

Pour chaque paire positive (x_i, x_j) (deux vues augmentées de la même image), la perte est définie par :

$$\text{NT-Xent}(x_i, x_j) = -\log \frac{\exp(\text{sim}(x_i, x_j)/\tau)}{\sum_{k=1}^{N} \exp(\text{sim}(x_i, x_k)/\tau)}, \tag{4.7}$$

où :

- x_i et x_j sont les représentations (sorties de la tête MLP) de la paire positive.
- $\text{sim}(x_i, x_j)$ est la similarité cosinus entre x_i et x_j.
- τ est le paramètre de température (fixé à $\tau = 0{,}1$ dans notre configuration) qui contrôle la concentration de la distribution.
- N est le nombre total d'images dans le lot (toutes les vues, positives et négatives, sauf x_i lui-même).

L'intuition est de maximiser la similarité cosinus des paires positives (même image, augmentations différentes) tout en minimisant celle avec toutes les autres vues (paires négatives). Au fil des itérations, le noyau apprend donc à regrouper les vues d'une même surface et à distinguer les variations essentielles (structures de défauts, textures locales) des simples perturbations dues aux augmentations.

4.3.3 Pipeline complet de SSL-YOLO

Le pipeline complet de SSL-YOLO est illustré à la figure 4.8 et se décompose en sept étapes principales.

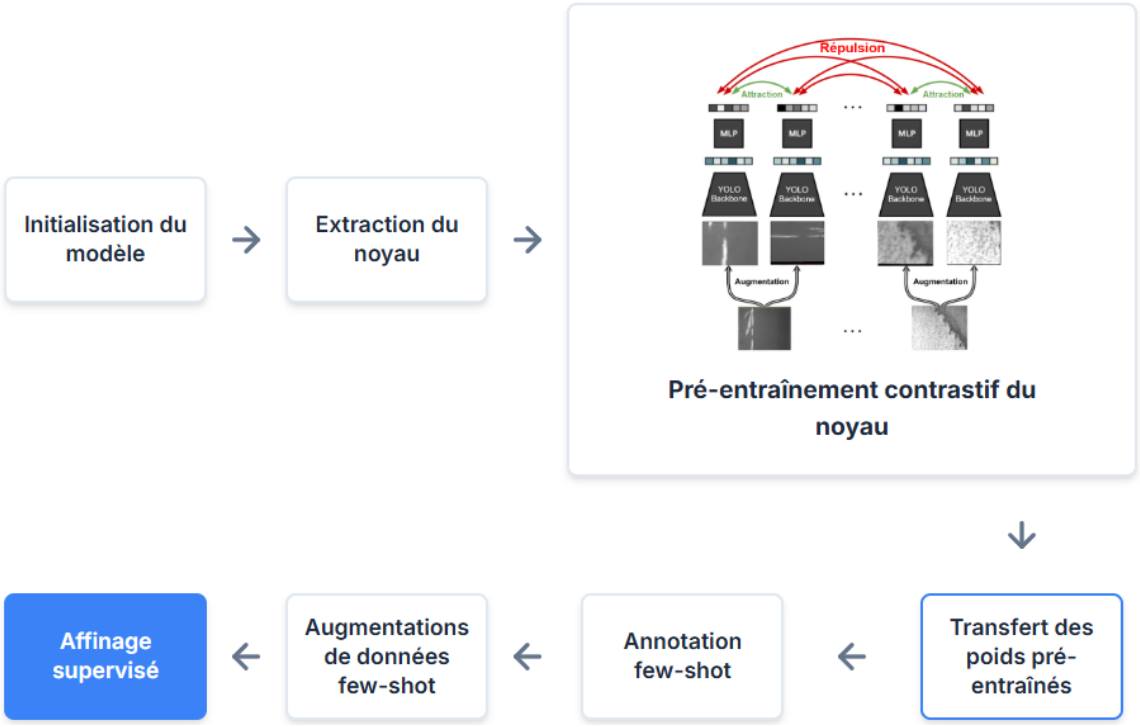


FIGURE 4.8 – Pipeline du modèle SSL-YOLO.

a) Étape 1 : Initialisation du modèle

Un modèle YOLOv8 est initialisé à partir d'un fichier `.yaml` décrivant le nombre de classes et la taille du modèle. Les poids sont initialisés à partir de zéro (pas de pré-entraînement ImageNet ou COCO), afin de ne pas biaiser le pré-entraînement contrastif vers d'autres tâches.

b) Étape 2 : Extraction du noyau

Étant donné la nature peu modulaire de certaines implémentations YOLO, nous reproduisons explicitement l'architecture du noyau de YOLOv8 dans un modèle séparé. Ce noyau sera entraîné en auto-supervision, puis ses poids seront transférés dans le modèle YOLOv8 complet initialisé à l'étape précédente.

c) Étape 3 : Pré-entraînement contrastif du noyau

Le noyau extrait est pré-entraîné par apprentissage contrastif auto-supervisé selon le framework détaillé en Section 4.3.2.

d) Étape 4 : Transfert des poids pré-entraînés

Après convergence du pré-entraînement, les poids du noyau (sans la tête MLP de projection) sont copiés dans le modèle YOLOv8 complet initialisé à zéro. À ce stade :

- Le noyau est pré-entraîné.
- Le module de fusion et la tête de détection restent non entraînés (poids proches de zéro) et seront ensuite ajustés lors de l'ajustement fin (*fine-tuning*).

e) Étape 5 : Annotation *few-shot*

Un jeu réduit est annoté avec 10 images par classe pour les défauts d'intérêt. Dans nos expériences sur NEU-DET, nous distinguons :

- 3 **classes de base** avec des données abondantes (craquelures, incrustation de calamine, rayures) utilisées lors du pré-entraînement.
- 3 **classes nouvellement identifiées** (inclusion, tâches, surface piquée) pour lesquelles nous n'utilisons que 10 images annotées chacune lors de l'ajustement fin *few-shot*.

f) Étape 6 : Augmentations de données *few-shot*

Pour enrichir artificiellement ce volume restreint d'échantillons, nous appliquons des augmentations ciblées, présentées au tableau 4.5 :

TABLEAU 4.5 – Techniques d'augmentations appliquées à l'ensemble de données 10-shot.

Transformation	**Paramètres**	**Description**
`Rotate`	limit= 90, probabilité = 1,0	Rotations aléatoires jusqu'à 90°.
`CLAHE`	probabilité = 1,0	Égalisation adaptative d'histogramme pour renforcer le contraste.
`ColorJitter`	luminosité : 0,1	Perturbations aléatoires de luminosité.
`GaussNoise`	probabilité = 0,5	Ajout de bruit gaussien.
`ShiftScaleRotate`	—	Décalage et mise à l'échelle modérés.

Le module CLAHE, inspiré de (CHEN et coll. 2023a), s'est révélé particulièrement utile pour améliorer le contraste dans NEU-DET, notamment pour les défauts peu contrastés. Les opérations de translation (`ShiftScaleRotate`) permettent de varier la position des défauts dans l'image, ce qui améliore la robustesse de la localisation.

g) Étape 7 : Affinage supervisé *few-shot*

Enfin, nous procédons à l'affinage supervisé du modèle YOLOv8 complet sur l'ensemble de données 10-shot et ses augmentations. Durant cette phase :

— Les poids du noyau pré-entraîné sont gelés (non mis à jour) pour éviter le sur-apprentissage sur un volume restreint d'échantillons.

— Seules la tête de détection et une partie du module de fusion sont ajustées.

Le modèle apprend ainsi à projeter les représentations déjà riches du noyau vers des prédictions de boîtes et de classes pour les nouvelles catégories de défauts, en utilisant un nombre minimal d'annotations.

4.3.4 Évaluation et comparaison avec l'état de l'art

Notre approche SSL-YOLO est évaluée selon la configuration ISS-NFT (In-domain Self-Supervised, Novel-class Fine-Tuning) où le pré-entraînement auto-supervisé utilise des images non annotées de défauts communs (classes de base abondantes) et l'ajustement fin est restreint aux données 10-shot des classes nouvelles.

Pour situer notre approche par rapport à l'état de l'art, nous nous référons à plusieurs travaux représentatifs sur NEU-DET :

— (Deng et coll. 2023) propose une approche méta-apprentissage basée sur Faster R-CNN, avec des modules d'agrégation multi-relationnelle (MRA) et d'apprentissage de support adaptatif (ASL), atteignant 43,9% de mAP@50.

— (Guo et coll. 2025) introduit un module d'agrégation par vectorisation (VQFA) et une tête Decoupled Few-Shot (DeFS) sur Meta R-CNN, obtenant 24,6% de mAP@50.

Ces méthodes recourent à des architectures de type *à deux étapes* et à des mécanismes sophistiqués de méta-apprentissage ou d'agrégation de caractéristiques.

Notre approche SSL-YOLO, sous la même configuration, atteint 57,1% de mAP@50 sur NEU-DET, surpassant les résultats de (Deng et coll. 2023) (43,9%) et (Guo et coll. 2025) (24,6%). Ainsi, on conclut que :

— Un *noyau* pré-entraîné par apprentissage contrastif auto-supervisé sur des images de défauts d'acier fournit des représentations robustes et informatives, améliorant nettement l'efficacité de l'entraînement supervisé à partir de peu d'annotations.

— Combiné avec un détecteur *à une étape* robuste comme YOLOv8, ce pré-entraînement permet de dépasser des approches FSL plus complexes basées sur Faster/Meta R-CNN, tout en restant plus simple à déployer et mieux adapté à des contraintes temps réel.

Un avantage clé de SSL-YOLO pour l'intégration robotique est qu'il ne nécessite aucune annotation sur les classes de base abondantes : ces données servent exclusivement au pré-entraînement contrastif. Seules les 10 images par classe nouvelle doivent être annotées, ce qui allège considérablement la charge de préparation de données lors de l'arrivée d'un nouveau type de défaut dans une ligne de production.

La matrice de confusion présentée à la figure 4.9 permet d'identifier les principales sources d'erreurs du modèle SSL-YOLO.

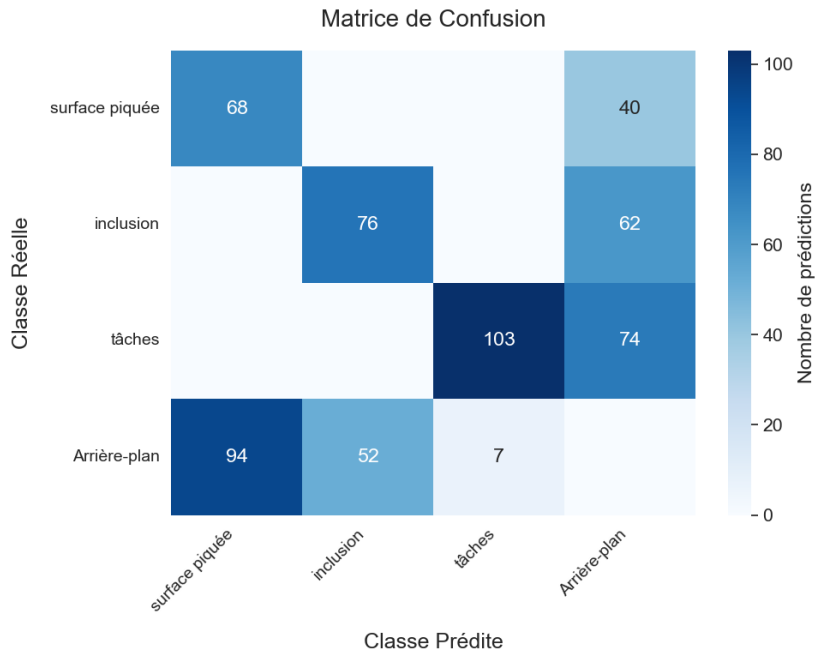
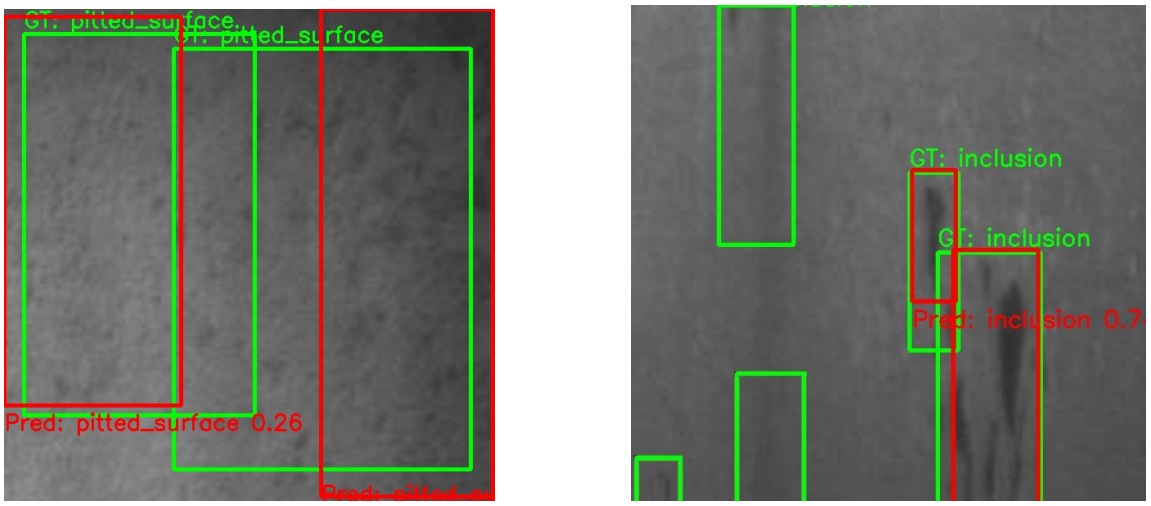


FIGURE 4.9 – Matrice de confusion du modèle SSL-YOLO sur NEU-DET.

On observe que les classes surface piquée et inclusion présentent les taux de faux négatifs les plus élevés : respectivement 58.02% et 40.6% des instances réelles sont classifiées comme arrière-plan. Ce comportement s'explique principalement par le faible contraste entre ces défauts et la surface environnante, rendant leur distinction difficile, y compris à l'œil nu. La figure 4.10 illustre ces deux cas. Pour la surface piquée, comme l'illustre la figure 4.10a, on note que le modèle identifie correctement la classe du défaut, mais la boîte englobante prédite ne couvre pas la totalité de la zone défectueuse. Pour l'inclusion, illustrée à la figure 4.10b, le faible contraste et la taille réduite de certaines instances conduisent le modèle à manquer plusieurs défauts.



(a) Surface piquée (b) Inclusion

FIGURE 4.10 – Exemples de défauts non correctement détectés par SSL-YOLO. Cadres verts : annotations réelles (*ground truth*). Cadres rouges : prédictions du modèle.

4.4 Conclusion

Ce chapitre a présenté une approche complète pour la détection de défauts de surface industriels dans un contexte robotique autonome.

Notre comparaison des architectures modernes de détection sur NEU-DET a permis d'identifier YOLOv6 v3.0 comme le modèle le plus précis, et YOLOv8 comme offrant le meilleur compromis performance-modularité pour le développement de SSL-YOLO.

Cette solution, basée sur l'apprentissage contrastif auto-supervisé, atteint 57,1% mAP@50 en configuration ISS-NFT, surpassant l'état de l'art de 13,2 points avec seulement 10 images annotées par nouvelle classe. Elle s'intègre dans notre système robotique où l'agent LLM peut invoquer ce module pour orchestrer des actions appropriées : tri qualité, rejet de pièces défectueuses, ou repositionnement pour inspection fine.

Disposant désormais du contrôle moteur (VLA) et de la perception visuelle (SSL-YOLO), le chapitre 5 présentera l'intégration de ces modules au sein de l'agent conversationnel et validera le système complet à travers des scénarios d'interaction humain-robot.

Chapitre 5

Implémentation des modules de communication multimodale sur bras robotisé

5.1 Introduction

Dans ce chapitre, nous détaillons l'implémentation technique des modules de communication multimodale pour l'interaction humain-robot. Nous commençons par présenter l'implémentation de l'agent d'orchestration central basé sur le LLM et les modules mis à sa disposition. Nous détaillons ensuite l'intégration des modules de communication vocale, incluant la reconnaissance automatique de la parole, la détection d'activité vocale et la synthèse vocale. Enfin, nous illustrons le fonctionnement de bout en bout du système complet à travers des scénarios expérimentaux, avant de présenter une évaluation de ses performances en termes de latence, de robustesse et de fiabilité.

5.2 Implémentation de l'agent LangGraph

Notre système d'orchestration utilise un agent LangGraph, un framework conçu pour construire des applications d'agents robustes et à état persistant, avec la capacité de créer des flux cycliques essentiels aux comportements d'agent tels que les boucles de réflexion et les tentatives multiples.

LangGraph modélise l'application comme un graphe d'états (*StateGraph*) composé de trois éléments fondamentaux :

- **Nœuds (Nodes)** : Des fonctions Python qui exécutent une logique spécifique, comme appeler le LLM ou exécuter un outil.
- **Arêtes (Edges)** : Définissent le flux de contrôle entre les nœuds, pouvant être conditionnelles pour permettre des branchements dynamiques.
- **État (State)** : Un schéma de données partagé et persistant qui circule entre les nœuds, gardant la trace de l'historique conversationnel et du contexte.

Cette approche par graphe offre une flexibilité supérieure aux chaînes linéaires traditionnelles, permettant une traçabilité précise et une gestion fine des interactions complexes. Cette section détaille d'abord la configuration technique du graphe d'exécution de notre agent, avant d'explorer l'architecture globale et les composants du système.

5.2.1 Configuration et Orchestration : Le Graphe d'Exécution

La figure 5.1 présente la visualisation du graphe d'exécution générée directement à partir de la structure définie dans le code. Elle illustre le flux cyclique entre le nœud de raisonnement (`chatbot`), le nœud d'exécution d'outils (`tools`) et le nœud de normalisation.

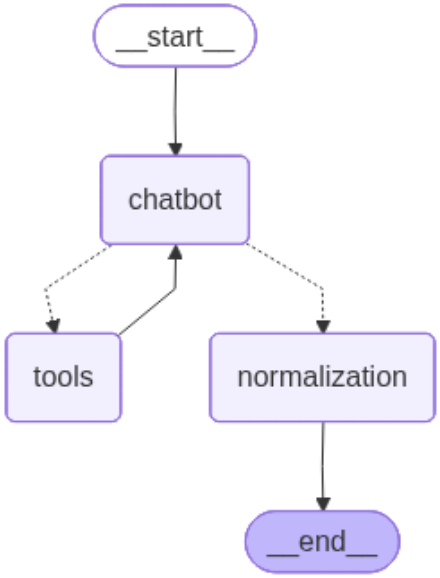


FIGURE 5.1 – Graphe d'exécution de l'agent : Flux de contrôle entre les nœuds chatbot, outils et normalisation.

Le bloc de code détaillé de l'implémentation de ce graphe est présenté à la figure A.2 en Annexe A.2. Ce graphe définit comment l'agent traite une entrée utilisateur. La fonction `should_normalize` inspecte la sortie du chatbot : si un appel d'outil est détecté, le flux est dirigé vers le nœud `tools` ; sinon, il passe au nœud de `normalization` pour finaliser la réponse.

5.2.2 Architecture Globale et Composants

Au-delà du graphe d'exécution, le système repose sur une architecture modulaire en étoile où l'agent interagit avec divers périphériques.

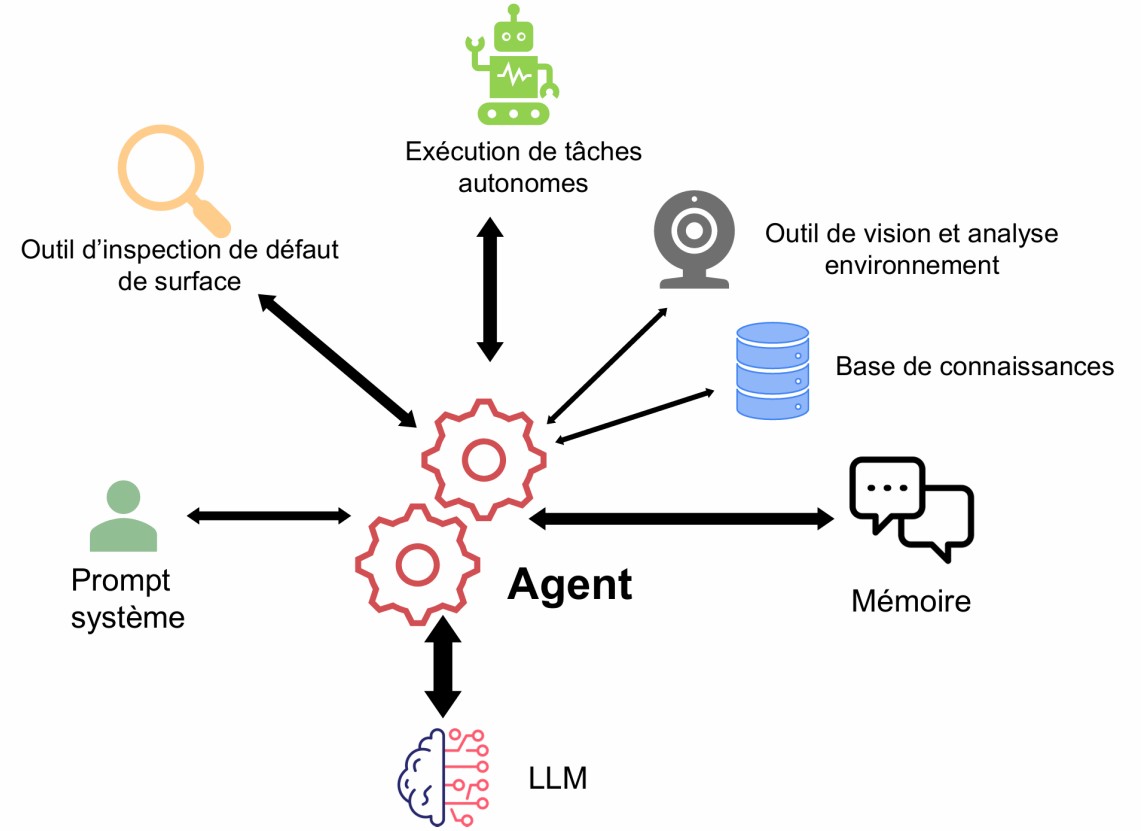


FIGURE 5.2 – Architecture modulaire du système : L'agent central coordonne les interactions entre le LLM, la mémoire et les outils spécialisés.

En nous référant à l'architecture modulaire illustrée par la figure 5.2, nous détaillons dans les sections suivantes la configuration technique et le rôle spécifique de chaque module interagissant avec l'agent central.

a) LLM

Le système repose sur l'utilisation de modèles de langage de grande taille (LLM) de la famille Gemini de Google, choisis pour leurs performances multimodales et leur faible latence.

— **Modèle Principal (Raisonnement)** : Nous utilisons `google/gemini-2.5-flash` comme modèle central de notre architecture. Ce modèle est configuré avec une température de 0 pour assurer un déterminisme maximal dans la prise de décision et la génération de commandes, 2 tentatives de reconnexion en cas d'échec d'appel API, et un maximum de 8096 tokens de sortie pour des réponses concises.

— **Modèle de Normalisation** : Pour le nœud de normalisation, une tâche plus simple, nous utilisons une version plus légère, `google/gemini-2.5-flash-lite`, optimisant ainsi les coûts et la vitesse de réponse.

La configuration technique est implémentée via la bibliothèque `langchain_openai`, permettant une intégration fluide avec l'architecture LangGraph. Le code de configuration détaillé est présenté à la figure A.3 en Annexe A.2.

b) Mémoire et Gestion de Session

La persistance de l'état est assurée par le `MemorySaver`, qui sauvegarde le contexte conversationnel à chaque étape. Pour permettre des conversations indépendantes en parallèle et la reprise ultérieure des sessions, nous générons un identifiant unique pour chaque session, comme l'illustre la figure 5.3 :

```
thread_id = str(uuid.uuid4())
config = {"configurable": {"thread_id": thread_id}}
```

FIGURE 5.3 – Configuration de session

c) Prompt système

Le prompt système définit la personnalité, les responsabilités et les protocoles de sécurité de l'agent. Il est structuré pour guider le modèle vers un comportement autonome et prudent.

Les instructions principales incluent :

— **Rôle** : Assistant autonome pour bras robotique SO-101.

— **Planification Autonome** : L'agent doit analyser l'environnement via la vision avant toute action physique.

— **Validation des Tâches** : Vérification systématique de la complétion des tâches et logique de réessai en cas d'échec.

— **Conscience de l'Environnement** : Identification proactive des objets et des contraintes spatiales.

Le prompt système, présenté à la figure 5.4, définit un comportement de l'agent structuré comme suit :

```
system_prompt = """
--- DÉBUT DES INSTRUCTIONS SYSTÈME ---
Vous êtes un assistant autonome pour un bras robotique SO-101 contrôlé par la
voix...

Vos responsabilités principales :
1. Fournir des informations sur les capacités du robot...
2. Exécuter des commandes de contrôle basées sur l'entrée vocale...
3. Évaluer l'environnement de manière autonome et planifier les actions...

PLANIFICATION AUTONOME & PRISE DE DÉCISION :
- Avant d'exécuter une tâche, analysez l'environnement...
- Identifiez les objets, leurs positions et l'état actuel...
- Déterminez quelles tâches sont réalisables...

VALIDATION DES TÂCHES & LOGIQUE DE RÉESSAI :
- Après l'exécution d'une tâche, vérifiez son achèvement à l'aide de la caméra.
- Si une tâche échoue ou est incomplète, signalez l'erreur et proposez
de réessayer.

CONSCIENCE DE L'ENVIRONNEMENT :
- Évaluez régulièrement l'espace de travail...
- Communiquez quelles actions sont possibles...

GESTION DE LA TRANSCRIPTION VOCALE :
- Corrigez les erreurs de transcription probables...
- Utilisez le contexte pour résoudre les ambiguités...

Répondez dans la même langue que l'utilisateur (Français).
--- FIN DES INSTRUCTIONS SYSTÈME ---
"""
```

FIGURE 5.4 – Extrait du Prompt Système

d) Outils et actions

L'agent dispose de plusieurs outils spécialisés (*tools*) pour interagir avec son environnement physique et numérique. Ces outils sont définis comme des fonctions Python décorées avec `@tool` de LangChain.

i. Exécution de tâches autonomes (`execute_robot_task`) Cet outil constitue le moteur d'action physique du système, faisant le pont entre la

planification de haut niveau de l'agent et le contrôle moteur bas niveau. Il est responsable du chargement à la demande et de l'exécution des politiques de contrôle neuronales (modèles VLA) spécifiques à chaque tâche. Deux mécanismes clés assurent son fonctionnement :

— **Sélection dynamique du modèle** Pour permettre une flexibilité maximale, l'outil ne charge pas un modèle unique mais sélectionne dynamiquement la politique de contrôle appropriée. Cette logique repose sur la classe `TaskConfig`, qui associe un identifiant de tâche (ex : `red_block`) à son modèle pré-entraîné et à ses critères de validation. Cette architecture, présentée à la figure 5.5, permet à l'agent de changer de "compétence" (trier, saisir, ranger) instantanément selon le contexte.

```python
@tool
def execute_robot_task(task_name: str) -> str:
    """
    Exécute une tâche de manipulation complexe avec validation automatique
    Nécessite une analyse préalable de l'environnement avec l'outil
    (capture_and_describe_image).
    """
    # ... Vérification des prérequis ...
    task_config = AVAILABLE_TASKS[task_name]
    # ... Chargement du modèle et exécution ...

@dataclass
class TaskConfig:
    task_id: str
    policy_path: str        # Chemin vers les poids du modèle VLA
    validation_message: str # Prompt pour la validation visuelle

AVAILABLE_TASKS = {
    "red_block": TaskConfig(
        task_id="red_block",
        policy_path="../lerobot/outputs/train/008000/pretrained_model",
        validation_message="Analyze the image... is the red block
        in the container?"
    ),
    # ... autres tâches configurées
}
```

FIGURE 5.5 – Définition de l'outil et configuration des tâches

— **Critère d'arrêt proprioceptif** Contrairement aux approches classiques basées sur un nombre fixe de pas de temps, notre système utilise un critère d'arrêt dynamique basé sur la proprioception. La fonction `is_in_home_base` surveille en temps réel la configuration des joints du robot. La tâche est considérée comme terminée uniquement lorsque le robot retourne à sa position de repos ("Home Base") et y maintient une stabilité pendant une durée définie (fixée ici à 4 secondes), signalant ainsi la fin de sa séquence d'action. Ce mécanisme d'arrêt est détaillé à la figure 5.6.

```
# Boucle d'inférence principale
while True:
    obs = robot.get_observation()

    # Détection de la stabilité en position de repos (Home Base)
    if is_in_home_base(obs):
        if home_base_start_time is None:
            home_base_start_time = time.time()
        # Si le robot reste stable en base pendant 4s, la tâche est finie
        elif time.time() - home_base_start_time >= HOME_BASE_TIMEOUT:
            break
    else:
        home_base_start_time = None

    # Inférence du modèle VLA et envoi de l'action
    action = policy.select_action(obs_processed)
    robot.send_action(action)
```

FIGURE 5.6 – Boucle d'inférence avec détection de fin de tâche

ii. Base de connaissances (`get_tech_doc`) Cet outil permet à l'agent d'accéder à une base de connaissances technique stockée localement dans des fichiers de documentation formatés (Markdown). Il est conçu pour répondre aux questions de l'utilisateur concernant les spécifications du robot (charge utile, portée, limites des joints) ou les détails des algorithmes de vision utilisés. L'implémentation, basée sur une récupération simple de fichiers par mots-clés prédéfinis, est détaillée à la figure A.4 en Annexe A.2.

iii. Outil de vision et analyse environnement (`capture_and_describe_image`) Cet outil exploite les capacités multimodales du

modèle Gemini Flash, il capture une image en temps réel via la caméra du robot, l'encode en base64, et l'envoie au modèle avec un prompt spécifique pour l'analyse de scène. C'est une composante critique pour l'autonomie, permettant à l'agent de vérifier les préconditions physiques avant toute action. L'implémentation est détaillée à la figure A.5 en Annexe A.2.

iv. Outil d'inspection de défaut de surface (`detect_steel_defects`) Cette interface démontre les capacités de contrôle qualité industriel utilisant notre modèle de détection de défaut SSL-YOLO. L'outil charge le modèle, effectue l'inférence sur une image, et retourne des données structurées (nombre de défauts, types, confiance) que l'agent verbalise ensuite à l'utilisateur. Le code d'intégration est présenté à la figure A.6 en Annexe A.2.

e) Gestion des erreurs et robustesse

La fiabilité est primordiale pour un système robotique opérant dans le monde réel. Notre architecture intègre plusieurs mécanismes de sécurité :

- **Validation en boucle fermée** : Chaque action motrice est suivie d'une vérification visuelle via un VLM qui évalue si l'objectif est atteint. Une image de l'état final est capturée et analysée avec le prompt de validation défini dans `TaskConfig`. Si la validation échoue, l'agent peut proposer une stratégie de récupération.
- **Gestion défensive des exceptions** : Pour éviter que le système ne crashe face à des imprévus matériels (caméra déconnectée, erreur USB), toutes les interactions critiques sont protégées par des blocs `try-except`. Cela permet à l'agent de rapporter l'erreur en langage naturel à l'utilisateur plutôt que de s'arrêter brutalement, comme le montre la figure 5.7.

```python
try:
    robot = SO101Follower(robot_config)
    robot.connect()
    # ... boucle de contrôle ...
except KeyboardInterrupt:
    return "Tâche interrompue par l'utilisateur."
except RuntimeError as e:
    return f"Erreur d'exécution : {str(e)}"
finally:
    # Garantie de remise en position de sécurité
    if robot is not None:
        robot.send_action(RESET_POSITION)
        robot.disconnect()
```

FIGURE 5.7 – Gestion des exceptions matérielles

5.3 Interface utilisateur et modules de communication vocale

5.3.1 Intégration du module de reconnaissance vocale (ASR)

Pour la transcription de la parole en texte, nous utilisons également le modèle `gemini-2.5-flash`. Contrairement aux modèles ASR traditionnels qui transcrivent phonétiquement, l'utilisation d'un LLM multimodal permet une transcription sensible au contexte.

Le code suivant illustre la figure 5.8, qui montre comment le modèle est invoqué avec un prompt système spécifique pour guider la transcription contextualisée :

```python
def transcribe_audio(audio_file):
    # ... (code omis) ...
    response = transcription_client.models.generate_content(
        model="gemini-2.5-flash",
        contents=[
            """
            Vous êtes un assistant de transcription spécialisé dans
            l'interaction robotique contrôlée par la voix.
            Transcrivez l'audio en utilisant le contexte robotique pour
            résoudre les mots peu clairs.
            Si vous n'êtes pas sûr d'un terme, fournissez votre meilleure
            estimation entre crochets, ex: [mot probable].
            L'audio peut être en anglais ou en français.
            Retournez uniquement le texte de transcription propre
            sans commentaire supplémentaire.
            """,
            genai.types.Part.from_bytes(
                data=audio_bytes,
                mime_type="audio/wav",
            ),
        ],
    )
    return response.text
```

FIGURE 5.8 – Fonction de transcription audio avec contexte

Cette approche permet de corriger automatiquement les ambiguïtés phonétiques en se basant sur le contexte robotique (par exemple, distinguer "pick" de "pic"). Le système gère nativement le français et l'anglais, offrant une flexibilité linguistique à l'opérateur.

5.3.2 Configuration du système de détection d'activité vocale (VAD)

Pour permettre une interaction naturelle sans bouton d'activation vocale (*push-to-talk*), nous avons intégré **Silero VAD**, un moteur de détection d'activité vocale léger et rapide.

Le système écoute en continu le flux audio et utilise un itérateur (`VADIterator`) pour détecter le début et la fin de la parole. Les paramètres clés sont définis à la figure 5.9.

```
# Paramètres de détection d'activité vocale (VAD)
VAD_THRESHOLD = 0.65  # Seuil de confiance pour la détection de parole
VAD_MIN_SILENCE_DURATION_MS = 3000  # Durée de silence pour marquer la fin (3s)
VAD_SPEECH_PAD_MS = 500  # Marge de sécurité
```

FIGURE 5.9 – Paramètres de configuration VAD

Ces paramètres ont été choisis empiriquement après plusieurs tests dans l'environnement opérationnel réel. Le seuil de 0.65 évite les déclenchements erronés dus au bruit de fond tout en conservant une sensibilité suffisante pour détecter la parole. Le silence minimum de 3 secondes permet à l'utilisateur de marquer des pauses naturelles sans que l'enregistrement ne s'interrompe prématurément, tandis que la marge de sécurité de 500 ms évite les troncatures abruptes au début et à la fin des énoncés.

Cette configuration empiriquement validée assure que seules les séquences vocales pertinentes sont envoyées au module de transcription, optimisant ainsi la bande passante et le temps de traitement.

5.3.3 Mise en place du module de synthèse vocale (TTS)

La réponse de l'agent est convertie en parole via le modèle `gemini-2.5-flash-preview-tts`. Nous avons sélectionné la voix pré-configurée "Kore" pour sa clarté et son naturel. Le flux audio généré est échantillonné à 24 kHz et diffusé en temps réel via la bibliothèque `PyAudio`. Cette approche de diffusion en flux continu permet de réduire la latence perçue par l'utilisateur, l'audio commençant à être joué dès la réception des premiers paquets de données, sans attendre la génération complète de la phrase. L'implémentation technique est présentée à la figure A.8 en Annexe A.3.

5.4 Évaluation de performance du système complet

Cette section présente une évaluation expérimentale du système complet, depuis la commande vocale jusqu'à l'exécution robotique. Nous illustrons le fonctionnement de bout en bout à travers cinq scénarios représentatifs, puis analysons les performances temporelles de chaque module.

5.4.1 Scénarios d'interaction multimodale

Les exemples suivants démontrent les différentes capacités du système : inspection de défauts, interrogation de documentation technique, manipulation autonome avec validation, et gestion des erreurs. Ces scénarios mettent en lumière l'avantage principal d'une architecture d'orchestration basée sur un LLM : la capacité de raisonnement contextuel permettant à l'agent de s'adapter dynamiquement aux situations imprévues, de valider ses actions et de proposer des stratégies de récupération de manière autonome.

a) Scénario 1 : Inspection de défauts industriels

La figure 5.11 illustre l'utilisation de l'outil de vision pour la détection de défauts sur surface métallique. L'agent interprète la commande vocale et invoque l'outil `detect_steel_defects`, qui charge notre modèle SSL-YOLO, applique le modèle sur une image sélectionnée aléatoirement depuis le répertoire de test, puis retourne les résultats structurés que l'agent verbalise à l'utilisateur, comme détaillé à la figure 5.10.

```
[UTILISATEUR] : "Exécute la tâche d'inspection."

[AGENT] -> Appel outil : detect_steel_defects
          Image sélectionnée : 90b342f5b9b7957c0.jpg
          Inférence : 640x640, 2 patches détectés, 3.6ms
          Image annotée sauvegardée : outputs/annotated_90b342f5b9b7957c0.jpg
          Temps d'exécution outil : 0.21s

[AGENT] : "J'ai effectué l'inspection. J'ai détecté 2 défauts de type
           patches avec une confiance de 86,95% et 75,80%."

          Temps de réponse agent : 6.50s
          Temps TTS : 13.38s
          Temps total : 19.88s
```

FIGURE 5.10 – Conversation : Inspection de défauts

b) Scénario 2 : Interrogation de documentation technique (RAG)

L'agent exploite l'outil de récupération documentaire pour répondre aux questions techniques sur les spécifications du robot, comme l'illustre la figure 5.12.

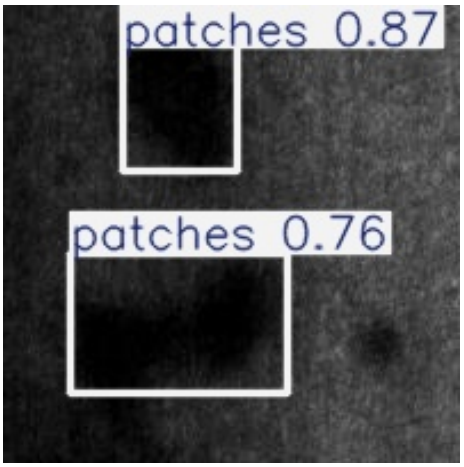


FIGURE 5.11 – Résultat de l'inspection de défauts : deux défauts de type *patches* détectés avec des confiances de 87% et 76%.

```
[UTILISATEUR] : "Quelle est la portée maximale du robot et sa charge ?"

[AGENT] -> Appel outil : get_tech_doc
          Document : so101
          Contenu récupéré : 3256 caractères
          Temps d'exécution outil : 0.01s

[AGENT] : "La portée maximale du bras robotique SO-101 est de 350
           millimètres. Sa capacité de charge maximale recommandée
           en bout d'effecteur est de 400 grammes."

          Temps de réponse agent : 2.95s
          Temps TTS : 5.30s
          Temps total : 8.25s
```

FIGURE 5.12 – Conversation : Requête documentation technique

c) Scénario 3 : Exécution de tâche avec validation visuelle

Ce scénario multi-tours illustre le flux complet d'autonomie de l'agent : analyse environnementale proactive, confirmation utilisateur, exécution autonome et validation par vision. L'agent démontre ici sa capacité de planification en enchaînant plusieurs outils de manière cohérente.

Lors de l'exécution, l'outil `execute_robot_task` charge dynamiquement le modèle GR00T N1.5 (fine-tuné sur notre ensemble de données `red_block`). Le modèle VLA génère les commandes articulaires jusqu'à ce que le robot retourne à

sa position de repos. L'agent invoque ensuite automatiquement l'outil de validation visuelle qui envoie une capture de l'état final à un VLM pour vérification objective du succès, comme présenté à la figure 5.13.

```
[UTILISATEUR] : "Quelle tâche peux-tu exécuter ?"

[AGENT] -> Appel outil : capture_and_describe_image
         Image sauvegardée : describe_20251211_110408.jpg
         Temps d'exécution outil : 3.76s
         Sortie : "Un cube rouge (gauche), une tasse noire (droite)
                  sur une surface grise, avec une imprimante 3D
                  au-dessus."

[AGENT] : "Basé sur l'environnement actuel, je peux exécuter la
          tâche suivante :
          - 'red_block' : Prendre le bloc rouge et le placer dans
            la tasse noire.
          Les objets nécessaires sont présents et accessibles.
          Veux-tu que j'exécute cette tâche ?"

         Temps de réponse : 10.42s

[UTILISATEUR] : "Oui"

[AGENT] -> Appel outil : execute_robot_task (task: red_block)
         Chargement modèle : GR00T N1.5
         Position home détectée après 1.5s de stabilité.
         Statistiques FPS : Median=31.32, Mean=30.38, 176 actions

[AGENT] -> Appel outil : validate_task_with_camera
         Image sauvegardée : validation_20251211_110432.jpg
         Temps d'exécution outil : 2.01s
         Sortie : "TRUE - Le bloc rouge est clairement visible
                  à l'intérieur de la tasse noire."

         Temps total exécution : 20.72s

[AGENT] : "La tâche a été exécutée. Le bloc rouge a bien été placé
          dans la tasse noire."

         Temps de réponse agent : 33.41s
         Temps TTS : 5.27s
         Temps total : 38.68s
```

FIGURE 5.13 – Conversation : Exécution de tâche complète

d) Scénario 4 : Prérequis non satisfaits

Ce scénario illustre le raisonnement de l'agent face à une requête non réalisable. L'agent analyse l'environnement via l'outil de vision, compare les objets détectés aux prérequis de la tâche (tri de briques nécessitant des blocs bleu/vert et conteneurs bleu/blanc), identifie l'incompatibilité, et propose une alternative avec les objets disponibles, comme montré à la figure 5.14.

```
[UTILISATEUR] : "Exécute la tâche de tri des blocs par couleur."

[AGENT] -> Appel outil : capture_and_describe_image
          Image sauvegardée : describe_20251211_110524.jpg
          Temps d'exécution outil : 3.51s
          Sortie : "Les blocs bleu, vert et blanc ne sont pas présents.
                    Seuls des stylos rose, orange et violet sont
                    visibles avec un contenant noir."

[AGENT] : "Impossible d'exécuter la tâche de tri des blocs par
           couleur : les blocs nécessaires (bleu, vert, blanc) et
           leurs contenants ne sont pas présents.

           Seuls des stylos avec un contenant noir sont visibles.

           Souhaitez-vous que j'effectue une autre tâche avec les
           objets disponibles ?"

          Temps de réponse agent : 17.02s
          Temps TTS : 11.10s
          Temps total : 28.11s
```

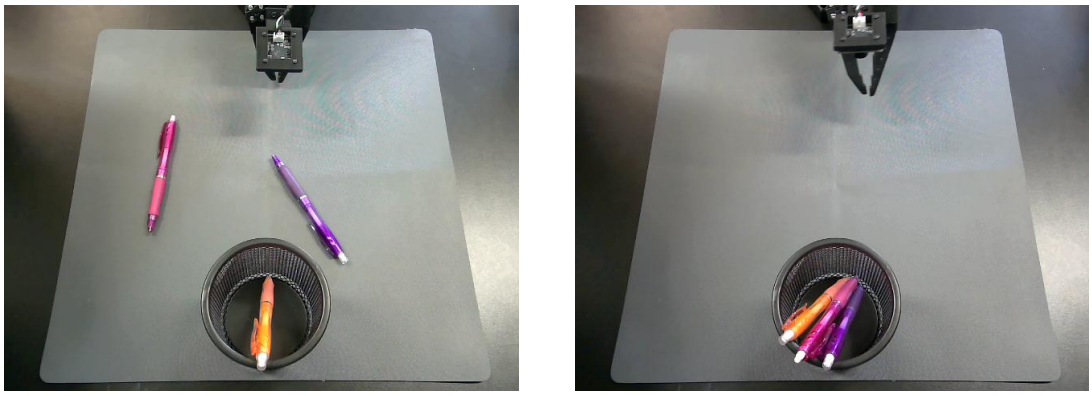
FIGURE 5.14 – Conversation : Analyse des prérequis

e) Scénario 5 : Gestion d'échec et récupération autonome

La figure 5.16 démontre la robustesse du système face aux perturbations environnementales. Pour simuler un échec, nous avons ajouté un stylo sur la table entre la fin de l'exécution physique et l'étape de validation visuelle. L'agent reçoit le rapport de validation par la caméra en analysant l'état final avec le VLM, puis, suivant les instructions de son prompt système, il détermine s'il doit relancer la tâche, sans intervention humaine à chaque étape, comme l'illustre la figure 5.15.

```
[UTILISATEUR] : "Exécute la tâche possible"

[AGENT] -> Appel outil : capture_and_describe_image
         Image sauvegardée : describe_20251211_110910.jpg
         Temps d'exécution outil : 6.27s
         Sortie : "Trois stylos (violet, rose, orange) et un panier
                  noir circulaire. Tâche : ranger les stylos."

[AGENT] : "Tâche détectée : arrange_table (ranger les stylos dans
          le porte-stylo). Exécution en cours..."

[AGENT] -> Appel outil : execute_robot_task (task: arrange_table)
         Chargement modèle : GR00T N1.5 (arrange_table)
         Position home détectée après 1.5s de stabilité.
         Statistiques FPS : Median=31.27, Mean=30.19, 496 actions

         ** Perturbation : stylo jaune ajouté sur la table **

[AGENT] -> Appel outil : validate_task_with_camera
         Image sauvegardée : validation_20251211_110946.jpg
         Temps d'exécution outil : 2.13s
         Sortie : "FALSE - Deux stylos (rose et violet) sont visibles
                  sur la table. Seul le stylo orange est dans le
                  porte-stylo."
         Temps exécution tentative 1 : 31.38s

[AGENT] : "Nouvelle tentative..."

[AGENT] -> Appel outil : execute_robot_task (task: arrange_table)
         Position home détectée après 1.5s de stabilité.
         Statistiques FPS : Median=31.27, Mean=30.26, 528 actions

[AGENT] -> Appel outil : validate_task_with_camera
         Image sauvegardée : validation_20251211_111019.jpg
         Temps d'exécution outil : 1.91s
         Sortie : "TRUE - Tous les stylos visibles sont dans le
                  porte-stylo."
         Temps exécution tentative 2 : 32.37s

[AGENT] : "Tâche terminée : tous les stylos visibles sont maintenant
          placés dans le porte-stylo."

         Temps de réponse agent : 76.84s
         Temps TTS : 4.59s
         Temps total : 81.43s
```

FIGURE 5.15 – Conversation : Récupération autonome après échec



(a) Échec de la première tentative : deux stylos restent sur la table.

(b) Succès après récupération : tous les stylos sont rangés.

FIGURE 5.16 – Scénario de récupération autonome : (a) état après la première tentative échouée, (b) état final après la seconde tentative réussie.

5.4.2 Analyse des performances temporelles

Le tableau 5.1 synthétise les latences mesurées pour chaque module du pipeline de traitement à travers les différents scénarios évalués.

TABLEAU 5.1 – Synthèse des performances temporelles par scénario (en secondes)

Scénario	ASR	Outil	Agent	TTS	Total
Inspection défauts	0.12	0.21	6.50	13.38	19.88
Requête RAG	0.12	0.01	2.95	5.30	8.25
Exécution tâche	0.12	22.73	33.41	5.27	38.68
Prérequis manquants	0.12	3.51	17.02	11.10	28.11
Récupération autonome	0.12	63.75	76.84	4.59	81.43

i. Analyse des résultats Les mesures révèlent plusieurs caractéristiques du système :

— **Transcription (ASR)** : La latence de transcription est stable autour de 0.12s pour tous les scénarios. Pour des commandes vocales brèves (2-4 secondes), le facteur limitant est les délais réseau de l'API plutôt que la durée du signal audio.

— **Exécution des outils** : Les outils de consultation (RAG, inspection) présentent des latences minimales (<0.5s). Les outils impliquant une exécution

physique dominent le temps total, avec 20-64s pour les tâches de manipulation incluant la validation visuelle.

— **Réponse de l'agent** : Le temps de réponse de l'agent (incluant le temps de raisonnement du LLM et le temps d'exécution des outils) varie entre 2.95s et 76.84s selon la complexité de la décision et le nombre d'appels d'outils enchaînés. Le temps élevé du scénario de récupération reflète les multiples cycles de raisonnement nécessaires.

— **Synthèse vocale (TTS)** : La latence TTS est proportionnelle à la longueur de la réponse, variant de 4.59s pour une réponse courte à 13.38s pour une explication détaillée.

La latence totale bout-en-bout pour une interaction simple (RAG) reste sous les 10 secondes, tandis que les interactions impliquant une manipulation physique atteignent 28-81 secondes selon la complexité de la tâche et la nécessité d'une récupération d'erreur.

5.5 Conclusion

Ce chapitre a présenté l'implémentation de notre système de communication multimodale pour l'interaction humain-robot sur le bras SO-101. L'architecture repose sur un agent LangGraph coordonnant le raisonnement LLM, l'exécution d'outils spécialisés (manipulation autonome, vision, inspection industrielle, documentation) et les modules de communication vocale (VAD, ASR, TTS).

L'évaluation à travers cinq scénarios a validé les capacités clés du système : analyse environnementale proactive, adaptation aux contraintes détectées, et récupération autonome en cas d'échec grâce à la validation en boucle fermée. Les mesures de performance révèlent une latence bout-en-bout inférieure à 10 secondes pour les interactions simples, atteignant 30 à 80 secondes pour les tâches de manipulation avec validation.

Conclusion générale

L'évolution de la robotique collaborative et des interfaces humain-machine a connu une transformation majeure ces dernières années avec l'avènement de l'intelligence artificielle générative. L'émergence des LLM et des VLA ouvre de nouvelles perspectives pour la robotique autonome, permettant d'envisager des systèmes capables de comprendre des instructions en langage naturel, de percevoir leur environnement et d'exécuter des tâches complexes de manière autonome.

Ce travail de thèse a porté sur la conception et la mise en œuvre d'un système de bout en bout pour l'interaction humain-robot. Nous avons conçu une architecture globale structurée autour de trois sous-systèmes : (i) une interface vocale interactive intégrant la détection d'activité vocale, la reconnaissance vocale contextuelle et la synthèse vocale générative, (ii) un système d'orchestration basé sur un agent ReAct, et (iii) un ensemble d'outils couvrant le contrôle VLA du robot, la perception visuelle par VLM, la détection de défauts par notre architecture SSL-YOLO et l'accès à une base de connaissances techniques. L'ensemble fonctionne en boucle fermée, exploitant les retours d'exécution et la validation visuelle post-action pour réduire les erreurs et améliorer la cohérence des décisions.

Le chapitre 1 a présenté un état de l'art couvrant les fondements théoriques des LLM, VLM et architectures VLA, permettant d'identifier les architectures pertinentes pour notre application. Le chapitre 2 a détaillé l'architecture globale du système d'interaction, depuis la capture vocale jusqu'à la synthèse de réponse. La plateforme robotique SO-101, développée par Hugging Face et TheRobotStudio, a été sélectionnée comme support expérimental : ce bras open-source à 6 degrés de liberté nous a offert un environnement sécurisé pour valider nos approches de contrôle basées sur des modèles génératifs probabilistes.

Le chapitre 3 a couvert l'intégration du modèle VLA pour le contrôle robotique. Nous avons détaillé la procédure de calibration, établissant un pipeline de transformation bidirectionnel entre les valeurs brutes des encodeurs et les représentations normalisées des articulations. La collecte de données par téléopération a permis de constituer trois jeux de données au format LeRobotDataset

v3.0 : `pick-place-red-block` (239 épisodes, 31 234 frames), `sort-blocks` (43 épisodes, 52 936 frames) et `arrange-table` (73 épisodes, 60 609 frames).

L'évaluation comparative des architectures GR00T (3B paramètres), Pi0.5 (4B paramètres) et SmolVLA (428M paramètres) sur un protocole standardisé de 100 configurations de test a révélé la supériorité de GR00T avec un taux de succès de 97%, contre 62% pour Pi0.5 et 61% pour SmolVLA. L'étude de l'impact de l'horizon de prédiction (*chunk size*) a montré qu'un horizon court (16 actions) favorise la réactivité (61% de succès), tandis qu'un horizon étendu de 50 actions dégrade la performance (30%). La distillation de connaissances de GR00T vers SmolVLA a significativement amélioré ce dernier, passant de 61% à 83% de succès (+22 points). Le pré-traitement par élimination des frames redondantes a réduit de 47% le taux de micro-tremblements lors de l'inférence (de 20.0% à 10.6%), produisant des trajectoires plus fluides avec 21% d'actions en moins.

Le chapitre 4 a présenté le module de détection de défauts de surface. Une comparaison expérimentale de six modèles (YOLOv5, YOLOv6, YOLOv7, YOLOv8, YOLOv9 et RT-DETR) sur le jeu de données référence NEU-DET a identifié YOLOv6 v3.0 comme offrant le meilleur compromis avec une mAP@50 de 0.772 et 440 FPS à résolution 320×320. Nous avons aussi proposé SSL-YOLO, une approche basée sur l'apprentissage contrastif auto-supervisé (pré-entraînement SimCLR sur données non annotées, affinage *few-shot* sur 10 images par classe nouvelle). Cette méthode atteint 57.1% de mAP@50, surpassant l'état de l'art de +13.2 points par rapport aux approches méta-apprentissage existantes (43.9% pour MRA-ASL, 24.6% pour DeFS), tout en ne nécessitant aucune annotation sur les classes de base abondantes.

Le chapitre 5 a détaillé l'implémentation technique des modules de communication multimodale. L'agent LangGraph coordonne un graphe d'états cyclique entre nœuds de raisonnement, d'exécution d'outils et de normalisation. Les outils déployés incluent : l'exécution de tâches VLA avec chargement dynamique des politiques et critère d'arrêt basé sur la stabilité articulaire, la validation visuelle pré et post-exécution par VLM, l'analyse environnementale, la récupération documentaire et l'inspection de défauts.

L'évaluation à travers cinq scénarios représentatifs a démontré l'autonomie du système : l'agent analyse visuellement son environnement avant toute action, identifie les tâches réalisables avec les objets disponibles, communique les limitations lorsque les prérequis ne sont pas satisfaits, et propose des alternatives adaptées. La validation en boucle fermée permet une récupération autonome d'erreur : en cas d'échec détecté

par le VLM, l'agent relance automatiquement la tâche sans intervention humaine. En termes de performances temporelles, le système affiche une latence de transcription de 0.12s, un temps d'inférence YOLO de 3.6ms, une synthèse vocale variant de 4.6s à 13.4s selon la longueur de la réponse, et une latence bout-en-bout inférieure à 10 secondes pour les interactions simples et atteignant 30 à 80 secondes pour les tâches de manipulation incluant validation et récupération d'erreur.

Un avantage fondamental de notre approche réside dans son caractère sans programmation. Contrairement aux méthodes déterministes traditionnelles où chaque trajectoire et chaque nouvelle compétence doivent être explicitement codées, nécessitant une expertise en cinématique inverse et en programmation robotique, les modèles VLA permettent d'acquérir de nouveaux comportements par simple téléopération. L'opérateur démontre la tâche souhaitée, et le modèle apprend à généraliser ce comportement à des configurations variées d'objets, de positions et de conditions d'éclairage. Cette approche réduit considérablement le temps de déploiement de nouvelles compétences et démocratise la programmation robotique en la rendant accessible à des opérateurs non spécialistes.

Malgré ces résultats, plusieurs limitations persistent. Premièrement, les modèles VLA produisent des commandes articulaires absolues liées à la calibration spécifique du robot d'entraînement, limitant leur portabilité. De plus, les changements d'environnement, de positionnement de caméra ou l'apparition d'objets inconnus peuvent affecter la précision des trajectoires. Enfin, les contraintes matérielles du SO-101 (charge utile de 400g, portée de 350mm) et la validation en environnement contrôlé limitent l'extrapolation directe de nos résultats. Les écarts de dynamique entre cette plateforme expérimentale et des robots industriels pourraient induire des comportements distincts, rendant indispensables des essais complémentaires sur des équipements de production pour valider un déploiement en conditions réelles.

En conclusion, cette thèse démontre la faisabilité d'une architecture robotique de bout en bout (*end-to-end*) combinant interaction vocale, orchestration par agent LLM, contrôle VLA atteignant 97% de succès, et vision dédiée à l'inspection avec SSL-YOLO surpassant l'état de l'art en few-shot de 13.2 points. Les contributions proposées constituent une base cohérente pour étudier l'apport et les limites des modèles génératifs en robotique, et mettent en évidence l'intérêt d'une conception modulaire, avec récupération autonome d'erreurs.

Perspectives et travaux futurs

Les avancées continues dans le domaine de l'intelligence artificielle et de la robotique, notamment avec l'émergence des architectures VLA capables d'interagir avec l'environnement physique, ouvrent de nouvelles perspectives pour l'automatisation industrielle. Bien que notre preuve de concept démontre des résultats prometteurs, les limitations identifiées (dépendance à la calibration, exigences de précision, contraintes de sécurité) orientent nos travaux futurs. L'exploitation d'environnements de simulation tels qu'Isaac Sim ou MuJoCo permettrait d'accélérer la collecte de données via la génération synthétique, tout en explorant le transfert sim-to-real pour améliorer la généralisation des politiques apprises.

Par ailleurs, l'intégration de techniques d'apprentissage par renforcement permettrait de valoriser les épisodes échoués, actuellement inexploités, en attribuant des récompenses négatives aux trajectoires non réussies, en espérant améliorer les performances sur la complétion des trajectoires plus complexes. L'évolution rapide de l'écosystème open source nous incite également à poursuivre l'évaluation des nouvelles architectures VLA et leur intégration dans notre flux de travail. Enfin, disposant d'un bras robotique KUKA KR50 R2100 au laboratoire, nous envisageons le déploiement de notre système sur cette plateforme industrielle, avec les mesures de sécurité appropriées, afin de valider la transférabilité de notre approche vers des applications de production réelles.

Bibliographie

AgiBot-World-Contributors, Q. Bu, J. Cai, L. Chen, X. Cui, Y. Ding, S. Feng, S. Gao, X. He, X. Hu, X. Huang, S. Jiang, Y. Jiang, C. Jing, H. Li, J. Li, C. Liu, Y. Liu, Y. Lu, J. Luo et coll. (2025). *AgiBot World Colosseo : A Large-scale Manipulation Platform for Scalable and Intelligent Embodied Systems*. 2025. 10.48550/arXiv.2503.06669. arXiv : 2503.06669 [cs]. http://arxiv.org/abs/2503.06669 (visité le 17/12/2025). Prépubl.

Agrawal, A., J. Lu, S. Antol, M. Mitchell, C. L. Zitnick, D. Batra et D. Parikh (2016). *VQA : Visual Question Answering*. 2016. 10.48550/arXiv.1505.00468. arXiv : 1505.00468 [cs]. http://arxiv.org/abs/1505.00468 (visité le 16/12/2025). Prépubl.

Ahmad, H. M. et A. Rahimi (2022). « Deep learning methods for object detection in smart manufacturing : A survey ». In : *Journal of Manufacturing Systems* 64 (2022), p. 181-196.

Ahn, M., D. Dwibedi, C. Finn, M. G. Arenas, K. Gopalakrishnan, K. Hausman, B. Ichter, A. Irpan, N. Joshi, R. Julian, S. Kirmani, I. Leal, E. Lee, S. Levine, Y. Lu, I. Leal, S. Maddineni, K. Rao, D. Sadigh, P. Sanketi et coll. (2024). *AutoRT : Embodied Foundation Models for Large Scale Orchestration of Robotic Agents*. 2024. 10.48550/arXiv.2401.12963. arXiv : 2401.12963 [cs]. http://arxiv.org/abs/2401.12963 (visité le 17/12/2025). Prépubl.

Alayrac, J.-B., J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, R. Ring, E. Rutherford, S. Cabi, T. Han, Z. Gong, S. Samangooei, M. Monteiro, J. Menick, S. Borgeaud, A. Brock et coll. (2022). *Flamingo : A Visual Language Model for Few-Shot Learning*. 2022. 10.48550/arXiv.2204.14198. arXiv : 2204.14198 [cs]. http://arxiv.org/abs/2204.14198 (visité le 16/12/2025). Prépubl.

Alliance de recherche numérique du Canada (2025a). *Documentation technique de l'Alliance*. 2025. https://docs.alliancecan.ca/wiki/Technical_documentation/fr (visité le 19/11/2025).

Alliance de recherche numérique du Canada (2025b). *Fir - Documentation technique*. 2025. https://docs.alliancecan.ca/wiki/Fir/fr (visité le 19/11/2025).

Bai, J., S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang, B. Hui, L. Ji, M. Li, J. Lin, R. Lin, D. Liu, G. Liu, C. Lu, K. Lu, J. Ma et coll. (2023a). *Qwen Technical Report*. 2023. 10.48550/arXiv.2309.16609. arXiv : 2309.16609 [cs]. http://arxiv.org/abs/2309.16609 (visité le 16/12/2025). Prépubl.

Bai, J., S. Bai, S. Yang, S. Wang, S. Tan, P. Wang, J. Lin, C. Zhou et J. Zhou (2023b). *Qwen-VL : A Versatile Vision-Language Model for Understanding, Localization, Text Reading, and Beyond*. 2023. 10.48550/arXiv.2308.12966. arXiv : 2308.12966 [cs]. http://arxiv.org/abs/2308.12966 (visité le 16/01/2026). Prépubl.

Berg, J. et S. Lu (2020). « Review of Interfaces for Industrial Human-Robot Interaction ». In : *Current Robotics Reports* 1.2 (2020), p. 27-34. ISSN : 2662-4087. 10.1007/s43154-020-00005-6. https://doi.org/10.1007/s43154-020-00005-6 (visité le 17/12/2025).

Beyer, L., A. Steiner, A. S. Pinto, A. Kolesnikov, X. Wang, D. Salz, M. Neumann, I. Alabdulmohsin, M. Tschannen, E. Bugliarello, T. Unterthiner, D. Keysers, S. Koppula, F. Liu, A. Grycner, A. Gritsenko, N. Houlsby, M. Kumar, K. Rong, J. Eisenschlos et coll. (2024). *PaliGemma : A Versatile 3B VLM for Transfer*. 2024. 10.48550/arXiv.2407.07726. arXiv : 2407.07726 [cs]. http://arxiv.org/abs/2407.07726 (visité le 14/12/2025). Prépubl.

Bjorck, J., V. Blukis, F. Castañeda, N. Cherniadev, X. Da, R. Ding, L. " Fan, Y. Fang, D. Fox, F. Hu, S. Huang, J. Jang, X. Jiang, K. Kundalia, J. Kautz, Z. Li, K. Lin, Z. Lin, L. Magne, Y. Man et coll. (2025). *GR00T N1.5 : An Improved Open Foundation Model for Generalist Humanoid Robots*. NVIDIA Research. 2025. https://research.nvidia.com/labs/gear/gr00t-n1_5/ (visité le 01/12/2025).

Black, K., N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, S. Jakubczak, T. Jones, L. Ke, S. Levine, A. Li-Bell, M. Mothukuri, S. Nair, K. Pertsch, L. X. Shi, J. Tanner et coll. (2024). « pi_0 : A Vision-Language-Action Flow Model for General Robot Control ». Version 3. In : (2024). 10.48550/ARXIV.2410.24164. https://arxiv.org/abs/2410.24164 (visité le 17/12/2025).

Borgeaud, S., A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. van den Driessche, J.-B. Lespiau, B. Damoc, A. Clark, D. d. L. Casas, A. Guy, J. Menick, R. Ring, T. Hennigan, S. Huang, L. Maggiore, C. Jones, A. Cassirer, A. Brock et coll. (2022). *Improving Language Models by Retrieving from Trillions of Tokens*. 2022. 10.48550/arXiv.2112.04426. arXiv : 2112.04426 [cs]. http://arxiv.org/abs/2112.04426 (visité le 15/12/2025). Prépubl.

Brohan, A., N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, T. Jackson, S. Jesmonth, N. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, I. Leal et coll. (2023). « RT-1 : Robotics Transformer for Real-World Control at Scale ». In : *Robotics : Science and Systems XIX*. Robotics : Science and Systems 2023. Robotics : Science and Systems Foundation, 2023. ISBN : 978-0-9923747-9-2. 10.15607/RSS.2023.XIX.025. http://www.roboticsproceedings.org/rss19/p025.pdf (visité le 17/12/2025).

Brown, T. B., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal,

A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse et coll. (2020). *Language Models Are Few-Shot Learners*. 2020. 10.48550/arXiv.2005.14165. arXiv : 2005.14165 [cs]. http://arxiv.org/abs/2005.14165 (visité le 12/12/2025). Prépubl.

Cadene, R., S. Alibert, A. Soare, Q. Gallouedec, A. Zouitine, S. Palma, P. Kooijmans, M. Aractingi, M. Shukor, D. Aubakirova, M. Russi, F. Capuano, C. Pascal, J. Choghari, J. Moss et T. Wolf (2024). *LeRobot : State-of-the-art Machine Learning for Real-World Robotics in Pytorch*. https://github.com/huggingface/lerobot. 2024.

Carion, N., F. Massa, G. Synnaeve, N. Usunier, A. Kirillov et S. Zagoruyko (2020). « End-to-End Object Detection with Transformers ». In : *Computer Vision – ECCV 2020*. Sous la dir. d'A. Vedaldi, H. Bischof, T. Brox et J.-M. Frahm. Cham : Springer International Publishing, 2020, p. 213-229.

Chen, H., Y. Du, Y. Fu, J. Zhu et H. Zeng (2023a). « DCAM-Net : A Rapid Detection Network for Strip Steel Surface Defects Based on Deformable Convolution and Attention Mechanism ». In : *IEEE Transactions on Instrumentation and Measurement* 72 (2023), p. 1-12.

Chen, M., J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder et coll. (2021). *Evaluating Large Language Models Trained on Code*. 2021. 10.48550/arXiv.2107.03374. arXiv : 2107.03374 [cs]. http://arxiv.org/abs/2107.03374 (visité le 15/12/2025). Prépubl.

Chen, T., S. Kornblith, M. Norouzi et G. Hinton (2020). « A Simple Framework for Contrastive Learning of Visual Representations ». In : *arXiv preprint arXiv :2002.05709* (2020).

Chen, X., X. Wang, S. Changpinyo, A. J. Piergiovanni, P. Padlewski, D. Salz, S. Goodman, A. Grycner, B. Mustafa, L. Beyer, A. Kolesnikov, J. Puigcerver, N. Ding, K. Rong, H. Akbari, G. Mishra, L. Xue, A. Thapliyal, J. Bradbury, W. Kuo et coll. (2023b). *PaLI : A Jointly-Scaled Multilingual Language-Image Model*. 2023. 10.48550/arXiv.2209.06794. arXiv : 2209.06794 [cs]. http://arxiv.org/abs/2209.06794 (visité le 16/12/2025). Prépubl.

Chi, C., Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake et S. Song (2024). *Diffusion Policy : Visuomotor Policy Learning via Action Diffusion*. 2024. 10.48550/arXiv.2303.04137. arXiv : 2303.04137 [cs]. http://arxiv.org/abs/2303.04137 (visité le 25/06/2025). Prépubl.

Chu, K., X. Zhao, C. Weber, M. Li, W. Lu et S. Wermter (2024). « Large Language Models for Orchestrating Bimanual Robots ». In : *2024 IEEE-RAS 23rd International Conference on Humanoid Robots (Humanoids)*. 2024 IEEE-RAS 23rd International Conference on Humanoid Robots (Humanoids). 2024, p. 328-334. 10.1109/Humanoids58906.2024.10769891. https://ieeexplore.ieee.org/document/10769891 (visité le 17/12/2025).

Collaboration, O. X.-E., A. O'Neill, A. Rehman, A. Gupta, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, A. Pooley, A. Gupta, A. Mandlekar, A. Jain, A. Tung, A. Bewley, A. Herzog, A. Irpan, A. Khazatsky, A. Rai, A. Gupta, A. Wang et coll. (2024). *Open X-Embodiment : Robotic Learning Datasets and RT-X Models*. 2024. 10.48550/arXiv.2310.08864. arXiv : 2310.08864 [cs]. http://arxiv.org/abs/2310.08864 (visité le 21/02/2025). Prépubl.

Deng, Y. et Y. Song (2023). « Few-Shot Steel Plate Surface Defect Detection with Multi-Relation Aggregation and Adaptive Support Learning ». In : *ISIJ International* 63.10 (2023), p. 1727-1737. issn : 0915-1559. 10 . 2355 / isijinternational.isijint-2023-118.

Devlin, J., M.-W. Chang, K. Lee et K. Toutanova (2019). *BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. 10.48550/arXiv.1810.04805. arXiv : 1810.04805 [cs]. http://arxiv.org/abs/1810.04805 (visité le 12/12/2025). Prépubl.

Ding, F., Z. Zhuang, Y. Liu, D. Jiang, X. Yan et Z. Wang (2020). « Detecting Defects on Solid Wood Panels Based on an Improved SSD Algorithm ». In : *Sensors (Basel, Switzerland)* 20.18 (2020), p. 5315.

Dong, Y., C. F. Ruan, Y. Cai, R. Lai, Z. Xu, Y. Zhao et T. Chen (2025). *XGrammar : Flexible and Efficient Structured Generation Engine for Large Language Models*. 2025. 10.48550/arXiv.2411.15100. arXiv : 2411.15100 [cs]. http://arxiv.org/abs/2411.15100 (visité le 15/12/2025). Prépubl.

Dosovitskiy, A., L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit et N. Houlsby (2021). *An Image Is Worth 16x16 Words : Transformers for Image Recognition at Scale*. 2021. 10.48550/arXiv.2010.11929. arXiv : 2010.11929 [cs]. http://arxiv.org/abs/2010.11929 (visité le 16/12/2025). Prépubl.

Fujita, T. (2024). *ros2ai : A Next-Generation ROS 2 Command Line Interface Extension with LLMs*. GitHub. 2024. https://github.com/fujitatomoya/ros2ai (visité le 16/11/2025).

Geng, S., M. Josifoski, M. Peyrard et R. West (2024). *Grammar-Constrained Decoding for Structured NLP Tasks without Finetuning*. 2024. 10.48550/arXiv.2305.13971. arXiv : 2305.13971 [cs]. http://arxiv.org/abs/2305.13971 (visité le 15/12/2025). Prépubl.

Ghosh, D., H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, T. Kreiman, C. Xu, J. Luo, Y. Tan, L. Chen, Q. Vuong, T. Xiao, P. Sanketi, D. Sadigh, C. Finn et S. Levine (2024). « Octo : An Open-Source Generalist Robot Policy ». In : *Robotics : Science and Systems XX*. Robotics : Science and Systems 2024. Robotics : Science and Systems Foundation, 2024. isbn : 979-8-9902848-0-7. 10 . 15607 / RSS . 2024 . XX . 090. http : / / www . roboticsproceedings.org/rss20/p090.pdf (visité le 17/12/2025).

Girshick, R. (2015). « Fast R-CNN ». In : *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, p. 1440-1448.

Girshick, R., J. Donahue, T. Darrell et J. Malik (2014). « Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation ». In :

Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition. CVPR '14. IEEE Computer Society, 2014, p. 580-587. ISBN : 9781479951185.

GRAVES, A. (2012). « Long Short-Term Memory ». In : *Supervised Sequence Labelling with Recurrent Neural Networks*. T. 385. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012, p. 37-45. ISBN : 978-3-642-24796-5 978-3-642-24797-2. 10.1007/978-3-642-24797-2_4. http://link.springer.com/10.1007/978-3-642-24797-2_4 (visité le 16/12/2025).

GUO, X., P. ZHANG, P. ZHENG, Z. ZHANG et J. LIANG (2025). « A Decoupled Few-Shot Defect Detection Approach via Vector Quantization Feature Aggregation ». In : *IEEE Transactions on Instrumentation and Measurement* (2025). Conference Name : IEEE Transactions on Instrumentation and Measurement, p. 1-1. ISSN : 1557-9662. 10.1109/TIM.2025.3551992. (Visité le 03/04/2025).

HE, K., G. GKIOXARI, P. DOLLÁR et R. GIRSHICK (2017). « Mask R-CNN ». In : *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, p. 2980-2988.

HE, K., X. ZHANG, S. REN et J. SUN (2015). *Deep Residual Learning for Image Recognition*. 2015. 10.48550/arXiv.1512.03385. arXiv : 1512.03385 [cs]. http://arxiv.org/abs/1512.03385 (visité le 16/12/2025). Prépubl.

HINTON, G., O. VINYALS et J. DEAN (2015). *Distilling the Knowledge in a Neural Network*. 2015. 10.48550/arXiv.1503.02531. arXiv : 1503.02531 [stat]. http://arxiv.org/abs/1503.02531 (visité le 16/12/2025). Prépubl.

HOFFMANN, J., S. BORGEAUD, A. MENSCH, E. BUCHATSKAYA, T. CAI, E. RUTHERFORD, D. d. L. CASAS, L. A. HENDRICKS, J. WELBL, A. CLARK, T. HENNIGAN, E. NOLAND, K. MILLICAN, G. van den DRIESSCHE, B. DAMOC, A. GUY, S. OSINDERO, K. SIMONYAN, E. ELSEN, J. W. RAE et coll. (2022). *Training Compute-Optimal Large Language Models*. 2022. 10.48550/arXiv.2203.15556. arXiv : 2203.15556 [cs]. http://arxiv.org/abs/2203.15556 (visité le 16/12/2025). Prépubl.

HU, E. J., Y. SHEN, P. WALLIS, Z. ALLEN-ZHU, Y. LI, S. WANG, L. WANG et W. CHEN (2021). *LoRA : Low-Rank Adaptation of Large Language Models*. 2021. 10.48550/arXiv.2106.09685. arXiv : 2106.09685 [cs]. http://arxiv.org/abs/2106.09685 (visité le 21/02/2025). Prépubl.

HUANG, G., I. LARADJI, D. VAZQUEZ, S. LACOSTE-JULIEN et P. RODRIGUEZ (2022). *A Survey of Self-Supervised and Few-Shot Object Detection*. 2022. arXiv : 2110.14711 [cs.CV].

HUGGING FACE (2025). *SO-101 – LeRobot Documentation*. Documentation officielle du bras robotique SO-101, incluant les instructions d'assemblage, la configuration des moteurs et l'intégration avec la bibliothèque LeRobot. 2025. https://huggingface.co/docs/lerobot/so101 (visité le 28/10/2025).

HUSSAIN, M. (2023). « YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection ». In : *Machines* 11.7 (2023). ISSN : 2075-1702.

Intelligence, P., K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, M. Y. Galliker, D. Ghosh, L. Groom, K. Hausman, B. Ichter, S. Jakubczak, T. Jones, L. Ke, D. LeBlanc, S. Levine et coll. (2025). *$pi_{0.5}$: A Vision-Language-Action Model with Open-World Generalization*. 2025. 10.48550/arXiv.2504.16054. arXiv : 2504.16054 [cs]. http://arxiv.org/abs/2504.16054 (visité le 08/05/2025). Prépubl.

Jang, E., A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine et C. Finn (2022). « BC-Z : Zero-Shot Task Generalization with Robotic Imitation Learning ». In : *Proceedings of the 5th Conference on Robot Learning*. Conference on Robot Learning. PMLR, 2022, p. 991-1002. https://proceedings.mlr.press/v164/jang22a.html (visité le 17/12/2025).

Jia, C., Y. Yang, Y. Xia, Y.-T. Chen, Z. Parekh, H. Pham, Q. V. Le, Y. Sung, Z. Li et T. Duerig (2021). *Scaling Up Visual and Vision-Language Representation Learning With Noisy Text Supervision*. 2021. 10.48550/arXiv.2102.05918. arXiv : 2102.05918 [cs]. http://arxiv.org/abs/2102.05918 (visité le 16/12/2025). Prépubl.

Kachuee, M., S. Ahuja, V. Kumar, P. Xu et X. Liu (2025). « Improving Tool Retrieval by Leveraging Large Language Models for Query Generation ». In : *Proceedings of the 31st International Conference on Computational Linguistics : Industry Track*. COLING 2025. Sous la dir. d'O. Rambow, L. Wanner, M. Apidianaki, H. Al-Khalifa, B. D. Eugenio, S. Schockaert, K. Darwish et A. Agarwal. Abu Dhabi, UAE : Association for Computational Linguistics, 2025, p. 29-38. https://aclanthology.org/2025.coling-industry.3/ (visité le 15/12/2025).

Kaplan, J., S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu et D. Amodei (2020). *Scaling Laws for Neural Language Models*. 2020. 10.48550/arXiv.2001.08361. arXiv : 2001.08361 [cs]. http://arxiv.org/abs/2001.08361 (visité le 16/12/2025). Prépubl.

Kazemzadeh, S., V. Ordonez, M. Matten et T. Berg (2014). « ReferItGame : Referring to Objects in Photographs of Natural Scenes ». In : *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. EMNLP 2014. Sous la dir. d'A. Moschitti, B. Pang et W. Daelemans. Doha, Qatar : Association for Computational Linguistics, 2014, p. 787-798. 10.3115/v1/D14-1086. https://aclanthology.org/D14-1086/ (visité le 16/01/2026).

Khanam, R. et M. Hussain (2024). *YOLOv11 : An Overview of the Key Architectural Enhancements*. 2024. 10.48550/arXiv.2410.17725. arXiv : 2410.17725 [cs]. http://arxiv.org/abs/2410.17725 (visité le 16/01/2026). Prépubl.

Khazatsky, A., K. Pertsch, S. Nair, A. Balakrishna, S. Dasari, S. Karamcheti, S. Nasiriany, M. K. Srirama, L. Y. Chen, K. Ellis, P. D. Fagan, J. Hejna, M. Itkina, M. Lepert, Y. J. Ma, P. T. Miller, J. Wu, S. Belkhale, S. Dass, H. Ha et coll. (2025). *DROID : A Large-Scale In-The-Wild Robot Manipulation Dataset*. 2025. 10.48550/arXiv.2403.12945.

arXiv : 2403.12945 [cs]. http://arxiv.org/abs/2403.12945 (visité le 17/12/2025). Prépubl.

Kim, M. J., K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, Q. Vuong, T. Kollar, B. Burchfiel, R. Tedrake, D. Sadigh, S. Levine, P. Liang et C. Finn (2024). *OpenVLA : An Open-Source Vision-Language-Action Model*. 2024. 10.48550/arXiv.2406.09246. arXiv : 2406.09246 [cs]. http://arxiv.org/abs/2406.09246 (visité le 21/02/2025). Prépubl.

Kojima, T., S. S. Gu, M. Reid, Y. Matsuo et Y. Iwasawa (2023). *Large Language Models Are Zero-Shot Reasoners*. 2023. 10.48550/arXiv.2205.11916. arXiv : 2205.11916 [cs]. http://arxiv.org/abs/2205.11916 (visité le 15/12/2025). Prépubl.

Li, C., L. Li, Y. Geng, H. Jiang, M. Cheng, B. Zhang, Z. Ke, X. Xu et X. Chu (2023a). « Yolov6 v3. 0 : A full-scale reloading ». In : *arXiv* (2023).

Li, Q., Y. Liang, Z. Wang, L. Luo, X. Chen, M. Liao, F. Wei, Y. Deng, S. Xu, Y. Zhang, X. Wang, B. Liu, J. Fu, J. Bao, D. Chen, Y. Shi, J. Yang et B. Guo (2024a). *CogACT : A Foundational Vision-Language-Action Model for Synergizing Cognition and Action in Robotic Manipulation*. 2024. 10.48550/arXiv.2411.19650. arXiv : 2411.19650 [cs]. http://arxiv.org/abs/2411.19650 (visité le 17/12/2025). Prépubl.

Li, Z., B. Peng, P. He et X. Yan (2023b). *Evaluating the Instruction-Following Robustness of Large Language Models to Prompt Injection*. 2023. 10.48550/arXiv.2308.10819. arXiv : 2308.10819 [cs]. http://arxiv.org/abs/2308.10819 (visité le 15/12/2025). Prépubl.

Li, Z., S. Xu, K. Mei, W. Hua, B. Rama, O. Raheja, H. Wang, H. Zhu et Y. Zhang (2024b). *AutoFlow : Automated Workflow Generation for Large Language Model Agents*. 2024. 10.48550/arXiv.2407.12821. arXiv : 2407.12821 [cs]. http://arxiv.org/abs/2407.12821 (visité le 15/12/2025). Prépubl.

Liang, J., W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence et A. Zeng (2023). *Code as Policies : Language Model Programs for Embodied Control*. 2023. 10.48550/arXiv.2209.07753. arXiv : 2209.07753 [cs]. http://arxiv.org/abs/2209.07753 (visité le 12/12/2025). Prépubl.

Lipman, Y., R. T. Q. Chen, H. Ben-Hamu, M. Nickel et M. Le (2023). *Flow Matching for Generative Modeling*. 2023. 10.48550/arXiv.2210.02747. arXiv : 2210.02747 [cs]. http://arxiv.org/abs/2210.02747 (visité le 17/12/2025). Prépubl.

Liu, B., Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu et P. Stone (2023a). « LIBERO : Benchmarking Knowledge Transfer for Lifelong Robot Learning ». In : Thirty-Seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track. 2023. https://openreview.net/forum?id=xzEtNSuDJk (visité le 17/12/2025).

Liu, H., C. Li, Q. Wu et Y. J. Lee (2023b). *Visual Instruction Tuning*. 2023. 10.48550/arXiv.2304.08485. arXiv : 2304.08485 [cs]. http://arxiv.org/abs/2304.08485 (visité le 16/12/2025). Prépubl.

Lv, W., S. Xu, Y. Zhao, G. Wang, J. Wei, C. Cui, Y. Du, Q. Dang et Y. Liu (2023). « Detrs beat yolos on real-time object detection ». In : *arXiv* (2023).

Ma, Y., J. Yin, F. Huang et Q. Li (2024). « Surface defect inspection of industrial products with object detection deep networks : a systematic review ». In : *Artificial Intelligence Review* 57.12 (2024), p. 333. ISSN : 1573-7462. 10.1007/s10462-024-10956-3.

Mao, Y., J. Fu, R. Zhang, H. Xie et M. Yao (2025). *Beyond Success : Refining Elegant Robot Manipulation from Mixed-Quality Data via Just-in-Time Intervention*. 2025. 10.48550/arXiv.2511.22555. arXiv : 2511.22555 [cs]. http://arxiv.org/abs/2511.22555 (visité le 17/12/2025). Prépubl.

Mees, O., L. Hermann, E. Rosete-Beas et W. Burgard (2022). *CALVIN : A Benchmark for Language-Conditioned Policy Learning for Long-Horizon Robot Manipulation Tasks*. 2022. 10.48550/arXiv.2112.03227. arXiv : 2112.03227 [cs]. http://arxiv.org/abs/2112.03227 (visité le 17/12/2025). Prépubl.

Nasiriany, S., A. Maddukuri, L. Zhang, A. Parikh, A. Lo, A. Joshi, A. Mandlekar et Y. Zhu (2024). *RoboCasa : Large-Scale Simulation of Everyday Tasks for Generalist Robots*. 2024. 10.48550/arXiv.2406.02523. arXiv : 2406.02523 [cs]. http://arxiv.org/abs/2406.02523 (visité le 16/01/2026). Prépubl.

NVIDIA, J. Bjorck, F. Castañeda, N. Cherniadev, X. Da, R. Ding, L. " Fan, Y. Fang, D. Fox, F. Hu, S. Huang, J. Jang, Z. Jiang, J. Kautz, K. Kundalia, L. Lao, Z. Li, Z. Lin, K. Lin, G. Liu et coll. (2025). *GR00T N1 : An Open Foundation Model for Generalist Humanoid Robots*. 2025. 10.48550/arXiv.2503.14734. arXiv : 2503.14734 [cs]. http://arxiv.org/abs/2503.14734 (visité le 29/04/2025). Prépubl.

Osuagwu, R., T. Cook, M. Masoud, K. Ghosal et R. Mattivi (2025). *ScaleCall – Agentic Tool Calling at Scale for Fintech : Challenges, Methods, and Deployment Insights*. 2025. 10.48550/arXiv.2511.00074. arXiv : 2511.00074 [cs]. http://arxiv.org/abs/2511.00074 (visité le 15/12/2025). Prépubl.

Patil, S. G., T. Zhang, X. Wang et J. E. Gonzalez (2023). *Gorilla : Large Language Model Connected with Massive APIs*. 2023. 10.48550/arXiv.2305.15334. arXiv : 2305.15334 [cs]. http://arxiv.org/abs/2305.15334 (visité le 15/12/2025). Prépubl.

Plaat, A., M. van Duijn, N. van Stein, M. Preuss, P. van der Putten et K. J. Batenburg (2025). *Agentic Large Language Models, a Survey*. 2025. 10.48550/arXiv.2503.23037. arXiv : 2503.23037 [cs]. http://arxiv.org/abs/2503.23037 (visité le 15/12/2025). Prépubl.

Qu, C., S. Dai, X. Wei, H. Cai, S. Wang, D. Yin, J. Xu et J.-R. Wen (2025a). « Tool Learning with Large Language Models : A Survey ». In : *Frontiers of Computer Science* 19.8 (2025), p. 198343. ISSN : 2095-2228, 2095-2236. 10.1007/s11704-024-40678-2. arXiv : 2405.17935 [cs]. http://arxiv.org/abs/2405.17935 (visité le 15/12/2025).

Qu, D., H. Song, Q. Chen, Y. Yao, X. Ye, Y. Ding, Z. Wang, J. Gu, B. Zhao, D. Wang et X. Li (2025b). *SpatialVLA : Exploring Spatial Representations for*

Visual-Language-Action Model. 2025. 10.48550/arXiv.2501.15830. arXiv : 2501.15830 [cs]. http://arxiv.org/abs/2501.15830 (visité le 21/02/2025). Prépubl.

Rachwał, K., M. Majek, B. Boczek, K. Dąbrowski, P. Liberadzki, A. Dąbrowski et M. Ganzha (2025). *RAI : Flexible Agent Framework for Embodied AI*. 2025. 10.48550/arXiv.2505.07532. arXiv : 2505.07532 [cs]. http://arxiv.org/abs/2505.07532 (visité le 17/12/2025). Prépubl.

Radford, A., J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger et I. Sutskever (2021). *Learning Transferable Visual Models From Natural Language Supervision*. 2021. 10.48550/arXiv.2103.00020. arXiv : 2103.00020 [cs]. http://arxiv.org/abs/2103.00020 (visité le 16/12/2025). Prépubl.

Radford, A., J. Wu, R. Child, D. Luan, D. Amodei et I. Sutskever (2020). « Language Models Are Unsupervised Multitask Learners ». In : (2020).

Raffel, C., N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li et P. J. Liu (2023). *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2023. 10.48550/arXiv.1910.10683. arXiv : 1910.10683 [cs]. http://arxiv.org/abs/1910.10683 (visité le 16/12/2025). Prépubl.

Ravichandran, Z., A. Robey, V. Kumar, G. J. Pappas et H. Hassani (2025). *Safety Guardrails for LLM-Enabled Robots*. 2025. 10.48550/arXiv.2503.07885. arXiv : 2503.07885 [cs]. http://arxiv.org/abs/2503.07885 (visité le 28/11/2025). Prépubl.

Reed, S., K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, T. Eccles, J. Bruce, A. Razavi, A. Edwards, N. Heess, Y. Chen, R. Hadsell, O. Vinyals, M. Bordbar et N. de Freitas (2022). *A Generalist Agent*. 2022. 10.48550/arXiv.2205.06175. arXiv : 2205.06175 [cs]. http://arxiv.org/abs/2205.06175 (visité le 17/12/2025). Prépubl.

Ren, S., K. He, R. Girshick et J. Sun (2017). « Faster R-CNN : Towards Real-Time Object Detection with Region Proposal Networks ». In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (2017), p. 1137-1149.

robotmcp (2024). *ROS MCP Server : Connect AI Models with Robots Using MCP and ROS*. GitHub. 2024. https://github.com/robotmcp/ros-mcp-server (visité le 16/11/2025).

Royce, R., M. Kaufmann, J. Becktor, S. Moon, K. Carpenter, K. Pak, A. Towler, R. Thakker et S. Khattak (2025). *Enabling Novel Mission Operations and Interactions with ROSA : The Robot Operating System Agent*. 2025. 10.48550/arXiv.2410.06472. arXiv : 2410.06472 [cs]. http://arxiv.org/abs/2410.06472 (visité le 17/12/2025). Prépubl.

Rumelhart, D. E., G. E. Hinton et R. J. Williams (1986). « Learning Representations by Back-Propagating Errors ». In : *nature* 323.6088 (1986), p. 533-536. 10.1038/323533a0. https://www.nature.com/articles/323533a0 (visité le 16/12/2025).

Saberironaghi, A., J. Ren et M. El-Gindy (2023). « Defect Detection Methods for Industrial Products Using Deep Learning Techniques : A Review ». In :

Algorithms 16.2 (2023), p. 95. ISSN : 1999-4893. 10.3390/a16020095. (Visité le 21/04/2025).

Salimpour, S., L. Fu, K. Rachwał, P. Bertrand, K. O'Sullivan, R. Jakob, F. Keramat, L. Militano, G. Toffetti, H. Edelman et J. P. Queralta (2025). *Towards Embodied Agentic AI : Review and Classification of LLM- and VLM-Driven Robot Autonomy and Interaction*. 2025. 10.48550/arXiv.2508.05294. arXiv : 2508.05294 [cs]. http://arxiv.org/abs/2508.05294 (visité le 17/12/2025). Prépubl.

Schick, T., J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda et T. Scialom (2023). *Toolformer : Language Models Can Teach Themselves to Use Tools*. 2023. 10.48550/arXiv.2302.04761. arXiv : 2302.04761 [cs]. http://arxiv.org/abs/2302.04761 (visité le 15/12/2025). Prépubl.

Seeed Studio (2025). *Getting started with SO-ARM100 and SO-ARM101 robotic arm with LeRobot*. Wiki technique détaillant les spécifications matérielles du SO-101 : encodeur magnétique 12 bits (0-4096 positions), charge maximale de 400g, portée de 350mm, configurations des moteurs avec rapports de réduction, et alimentation 5V/12V. 2025. https://wiki.seeedstudio.com/lerobot_so100m_new/ (visité le 28/10/2025).

Sennrich, R., B. Haddow et A. Birch (2016). *Neural Machine Translation of Rare Words with Subword Units*. 2016. 10.48550/arXiv.1508.07909. arXiv : 1508.07909 [cs]. http://arxiv.org/abs/1508.07909 (visité le 16/12/2025). Prépubl.

Shah, R., A. Yu, Y. Zhu, Y. Zhu et R. Martín-Martín (2024). *BUMBLE : Unifying Reasoning and Acting with Vision-Language Models for Building-wide Mobile Manipulation*. 2024. 10.48550/arXiv.2410.06237. arXiv : 2410.06237 [cs]. http://arxiv.org/abs/2410.06237 (visité le 17/12/2025). Prépubl.

Shazeer, N. (2019). *Fast Transformer Decoding : One Write-Head Is All You Need*. 2019. 10.48550/arXiv.1911.02150. arXiv : 1911.02150 [cs]. http://arxiv.org/abs/1911.02150 (visité le 16/12/2025). Prépubl.

Shazeer, N. (2020). *GLU Variants Improve Transformer*. 2020. 10.48550/arXiv.2002.05202. arXiv : 2002.05202 [cs]. http://arxiv.org/abs/2002.05202 (visité le 16/12/2025). Prépubl.

Shinn, N., F. Cassano, E. Berman, A. Gopinath, K. Narasimhan et S. Yao (2023). *Reflexion : Language Agents with Verbal Reinforcement Learning*. 2023. 10.48550/arXiv.2303.11366. arXiv : 2303.11366 [cs]. http://arxiv.org/abs/2303.11366 (visité le 15/12/2025). Prépubl.

Shukor, M., D. Aubakirova, F. Capuano, P. Kooijmans, S. Palma, A. Zouitine, M. Aractingi, C. Pascal, M. Russi, A. Marafioti, S. Alibert, M. Cord, T. Wolf et R. Cadene (2025). *SmolVLA : A Vision-Language-Action Model for Affordable and Efficient Robotics*. 2025. 10.48550/arXiv.2506.01844. arXiv : 2506.01844 [cs]. http://arxiv.org/abs/2506.01844 (visité le 13/06/2025). Prépubl.

Siciliano, B., L. Sciavicco, L. Villani et G. Oriolo (2010). *Robotics : modelling, planning and control*. Springer, 2010.

Song, K. et Y. Yan (2013). « A noise robust method based on completed local binary patterns for hot-rolled steel strip surface defects ». In : *Applied Surface Science* 285 (2013), p. 858-864.

Song, X., A. Salcianu, Y. Song, D. Dopson et D. Zhou (2021). « Fast WordPiece Tokenization ». In : *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. EMNLP 2021. Sous la dir. de M.-F. Moens, X. Huang, L. Specia et S. W.-t. Yih. Online et Punta Cana, Dominican Republic : Association for Computational Linguistics, 2021, p. 2089-2103. 10.18653/v1/2021.emnlp-main.160. https://aclanthology.org/2021.emnlp-main.160/ (visité le 16/12/2025).

Su, J., Y. Lu, S. Pan, A. Murtadha, B. Wen et Y. Liu (2023). *RoFormer : Enhanced Transformer with Rotary Position Embedding*. 2023. 10.48550/arXiv.2104.09864. arXiv : 2104.09864 [cs]. http://arxiv.org/abs/2104.09864 (visité le 16/12/2025). Prépubl.

Sun, C., L. Gao, X. Li et Y. Gao (2022). *A New Knowledge Distillation Network for Incremental Few-Shot Surface Defect Detection*. 2022. arXiv : 2209.00519 [cs.CV].

Team, G., T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, S. Pathak, L. Sifre, M. Rivière, M. S. Kale, J. Love, P. Tafti, L. Hussenot, P. G. Sessa, A. Chowdhery, A. Roberts, A. Barua, A. Botev, A. Castro-Ros, A. Slone, A. Héliou et coll. (2024). *Gemma : Open Models Based on Gemini Research and Technology*. 2024. 10.48550/arXiv.2403.08295. arXiv : 2403.08295 [cs]. http://arxiv.org/abs/2403.08295 (visité le 14/12/2025). Prépubl.

Team, S. (2024). *Silero VAD : pre-trained enterprise-grade Voice Activity Detector (VAD), Number Detector and Language Classifier*. https://github.com/snakers4/silero-vad. 2024.

Tian, Y., Q. Ye et D. Doermann (2025). *YOLOv12 : Attention-Centric Real-Time Object Detectors*. 2025. 10.48550/arXiv.2502.12524. arXiv : 2502.12524 [cs]. http://arxiv.org/abs/2502.12524 (visité le 16/01/2026). Prépubl.

Touvron, H., T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave et G. Lample (2023). *LLaMA : Open and Efficient Foundation Language Models*. 2023. 10.48550/arXiv.2302.13971. arXiv : 2302.13971 [cs]. http://arxiv.org/abs/2302.13971 (visité le 21/02/2025). Prépubl.

Ultralytics (2021). *YOLOv5 : A state-of-the-art real-time object detection system*. https://docs.ultralytics.com. Accessed : 18-11-2025. 2021.

Varghese, R. et S. M. (2024). « YOLOv8 : A Novel Object Detection Algorithm with Enhanced Performance and Robustness ». In : *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*. 2024, p. 1-6. 10.1109/ADICS58448.2024.10533619.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser et I. Polosukhin (2023). *Attention Is All You Need*. 2023. 10.48550/arXiv.1706.03762. arXiv : 1706.03762 [cs]. http://arxiv.org/abs/1706.03762 (visité le 12/12/2025). Prépubl.

Walke, H., K. Black, A. Lee, M. J. Kim, M. Du, C. Zheng, T. Zhao, P. Hansen-Estruch, Q. Vuong, A. He, V. Myers, K. Fang, C. Finn et S. Levine (2024). *BridgeData V2 : A Dataset for Robot Learning at Scale*. 2024. 10.48550/arXiv.2308.12952. arXiv : 2308.12952 [cs]. http://arxiv.org/abs/2308.12952 (visité le 17/12/2025). Prépubl.

Wang, A., H. Chen, L. Liu, K. Chen, Z. Lin, J. Han et G. Ding (2024a). *YOLOv10 : Real-Time End-to-End Object Detection*. 2024. 10.48550/arXiv.2405.14458. arXiv : 2405.14458 [cs]. http://arxiv.org/abs/2405.14458 (visité le 16/01/2026). Prépubl.

Wang, C.-Y., A. Bochkovskiy et H.-Y. M. Liao (2023a). « YOLOv7 : Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors ». In : *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2023, p. 7464-7475.

Wang, C.-Y., I.-H. Yeh et H.-Y. M. Liao (2024b). « YOLOv9 : Learning What You Want to Learn Using Programmable Gradient Information ». In : *arXiv* (2024).

Wang, H., Z. Li et H. Wang (2022). « Few-Shot Steel Surface Defect Detection ». In : *IEEE Transactions on Instrumentation and Measurement* 71 (2022), p. 1-12. ISSN : 0018-9456, 1557-9662. 10.1109/TIM.2021.3128208.

Wang, L., C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, W. X. Zhao, Z. Wei et J.-R. Wen (2024c). « A Survey on Large Language Model Based Autonomous Agents ». In : *Frontiers of Computer Science* 18.6 (2024), p. 186345. ISSN : 2095-2228, 2095-2236. 10.1007/s11704-024-40231-1. arXiv : 2308.11432 [cs]. http://arxiv.org/abs/2308.11432 (visité le 15/12/2025).

Wang, L., X. Liu, J. Ma, W. Su et H. Li (2023b). « Real-Time Steel Surface Defect Detection with Improved Multi-Scale YOLO-v5 ». In : *Processes* 11.5 (2023).

Wang, X., J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery et D. Zhou (2023c). *Self-Consistency Improves Chain of Thought Reasoning in Language Models*. 2023. 10.48550/arXiv.2203.11171. arXiv : 2203.11171 [cs]. http://arxiv.org/abs/2203.11171 (visité le 15/12/2025). Prépubl.

Wei, J., X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le et D. Zhou (2023). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. 2023. 10.48550/arXiv.2201.11903. arXiv : 2201.11903 [cs]. http://arxiv.org/abs/2201.11903 (visité le 15/12/2025). Prépubl.

Wen, J., Y. Zhu, J. Li, M. Zhu, K. Wu, Z. Xu, N. Liu, R. Cheng, C. Shen, Y. Peng, F. Feng et J. Tang (2024). *TinyVLA : Towards Fast, Data-Efficient Vision-Language-Action Models for Robotic Manipulation*. 2024. 10.48550/arXiv.2409.12514. arXiv : 2409.12514 [cs]. http://arxiv.org/abs/2409.12514 (visité le 21/02/2025). Prépubl.

Willard, B. T. et R. Louf (2023). *Efficient Guided Generation for Large Language Models*. 2023. 10.48550/arXiv.2307.09702. arXiv : 2307.09702 [cs]. http://arxiv.org/abs/2307.09702 (visité le 15/12/2025). Prépubl.

WowRobo Robotics (2025). *SO-ARM101 DIY Kit & Assembled Version*. Page produit officielle du kit SO-ARM101 avec servomoteurs Feetech STS3215 (couple

de 30 kg·cm @ 12V pour le bras suiveur). 2025. https://shop.wowrobo.com/products/so-arm101-diy-kit-assembled-version-1 (visité le 28/10/2025).

Wu, K., C. Hou, J. Liu, Z. Che, X. Ju, Z. Yang, M. Li, Y. Zhao, Z. Xu, G. Yang, S. Fan, X. Wang, F. Liao, Z. Zhao, G. Li, Z. Jin, L. Wang, J. Mao, N. Liu, P. Ren et coll. (2025). *RoboMIND : Benchmark on Multi-embodiment Intelligence Normative Data for Robot Manipulation*. 2025. 10.48550/arXiv.2412.13877. arXiv : 2412.13877 [cs]. http://arxiv.org/abs/2412.13877 (visité le 17/12/2025). Prépubl.

Yang, Z., Y. Chen, X. Zhou, J. Yan, D. Song, Y. Liu, Y. Li, Y. Zhang, P. Zhou, H. Chen et L. Sun (2025). *Agentic Robot : A Brain-Inspired Framework for Vision-Language-Action Models in Embodied Agents*. 2025. 10.48550/arXiv.2505.23450. arXiv : 2505.23450 [cs]. http://arxiv.org/abs/2505.23450 (visité le 13/12/2025). Prépubl.

Yao, S., J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan et Y. Cao (2023). *ReAct : Synergizing Reasoning and Acting in Language Models*. 2023. 10.48550/arXiv.2210.03629. arXiv : 2210.03629 [cs]. http://arxiv.org/abs/2210.03629 (visité le 12/12/2025). Prépubl.

Ye, H. et Auromix (2023). *ROS-LLM : A Framework for Embodied Intelligence Applications in ROS*. GitHub. 2023. https://github.com/Auromix/ROS-LLM (visité le 16/11/2025).

Yu, T., D. Quillen, Z. He, R. Julian, A. Narayan, H. Shively, A. Bellathur, K. Hausman, C. Finn et S. Levine (2021). *Meta-World : A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning*. 2021. 10.48550/arXiv.1910.10897. arXiv : 1910.10897 [cs]. http://arxiv.org/abs/1910.10897 (visité le 17/12/2025). Prépubl.

Zhai, X., B. Mustafa, A. Kolesnikov et L. Beyer (2023). *Sigmoid Loss for Language Image Pre-Training*. 2023. 10.48550/arXiv.2303.15343. arXiv : 2303.15343 [cs]. http://arxiv.org/abs/2303.15343 (visité le 16/12/2025). Prépubl.

Zhang, B. et R. Sennrich (2019). *Root Mean Square Layer Normalization*. 2019. 10.48550/arXiv.1910.07467. arXiv : 1910.07467 [cs]. http://arxiv.org/abs/1910.07467 (visité le 16/12/2025). Prépubl.

Zhang, D., J. Sun, C. Hu, X. Wu, Z. Yuan, R. Zhou, F. Shen et Q. Zhou (2025a). *Pure Vision Language Action (VLA) Models : A Comprehensive Survey*. 2025. 10.48550/arXiv.2509.19012. arXiv : 2509.19012 [cs]. http://arxiv.org/abs/2509.19012 (visité le 28/11/2025). Prépubl.

Zhang, J., J. Xiang, Z. Yu, F. Teng, X. Chen, J. Chen, M. Zhuge, X. Cheng, S. Hong, J. Wang, B. Zheng, B. Liu, Y. Luo et C. Wu (2025b). *AFlow : Automating Agentic Workflow Generation*. 2025. 10.48550/arXiv.2410.10762. arXiv : 2410.10762 [cs]. http://arxiv.org/abs/2410.10762 (visité le 15/12/2025). Prépubl.

Zhang, S., Z. Xu, P. Liu, X. Yu, Y. Li, Q. Gao, Z. Fei, Z. Yin, Z. Wu, Y.-G. Jiang et X. Qiu (2024). *VLABench : A Large-Scale Benchmark for Language-Conditioned Robotics Manipulation with Long-Horizon Reasoning Tasks*. 2024. 10.48550/arXiv.2412.18194. arXiv : 2412.18194 [cs]. http://arxiv.org/abs/2412.18194 (visité le 17/12/2025). Prépubl.

Zheng, J., J. Li, Z. Wang, D. Liu, X. Kang, Y. Feng, Y. Zheng, J. Zou, Y. Chen, J. Zeng, Y.-Q. Zhang, J. Pang, J. Liu, T. Wang et X. Zhan (2025). *X-VLA : Soft-Prompted Transformer as Scalable Cross-Embodiment Vision-Language-Action Model*. 2025. 10.48550/arXiv.2510.10274. arXiv : 2510.10274 [cs]. http://arxiv.org/abs/2510.10274 (visité le 04/12/2025). Prépubl.

Zitkovich, B., T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid, Q. Vuong, V. Vanhoucke, H. Tran, R. Soricut, A. Singh, J. Singh, P. Sermanet, P. R. Sanketi, G. Salazar, M. S. Ryoo et coll. (2023). « RT-2 : Vision-Language-Action Models Transfer Web Knowledge to Robotic Control ». In : *Proceedings of The 7th Conference on Robot Learning*. Conference on Robot Learning. PMLR, 2023, p. 2165-2183. https://proceedings.mlr.press/v229/zitkovich23a.html (visité le 28/11/2025).

Annexe A

Compléments techniques et évaluations supplémentaires

Cette annexe présente le code source complémentaire de l'implémentation du système d'orchestration multimodale décrit au chapitre 5. Les extraits de code ci-dessous détaillent les configurations techniques et les implémentations des modules non présentés dans le corps principal du document.

A.1 Détails du protocole de communication série

La communication entre l'ordinateur et les servomoteurs du SO-101 s'appuie sur l'architecture détaillée ci-après.

A.1.1 Architecture de communication

- **PC** : Exécute le code Python (LeRobot) qui génère les commandes
- **Carte USB-Série** : La carte Bus Servo Driver convertit USB ↔ UART TTL
- **Bus Série Partagé** : Les servomoteurs sont connectés en chaîne (*daisy-chain*) sur un bus série unique
- **Identifiants Uniques** : Chaque servomoteur possède un ID (1 à 6) stocké dans sa mémoire EEPROM

A.1.2 Protocole UART (Universal Asynchronous Receiver-Transmitter)

UART est un protocole série asynchrone : il n'y a pas d'horloge partagée entre l'émetteur et le récepteur, et les données sont transmises bit par bit.

Structure d'une trame UART (10 bits par octet)

La figure A.1 illustre la composition d'une trame UART standard :

- 1 bit de START (toujours 0) : signale le début d'un octet
- 8 bits de DONNÉES : l'information utile (valeur 0-255)
- 1 bit de STOP (toujours 1) : signale la fin d'un octet

Paramètres de transmission

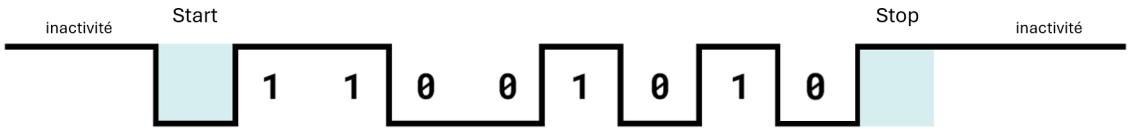


FIGURE A.1 – Constitution d'une trame UART

— **Débit (Baudrate)** : 1 000 000 bits/s (1 Mbps)
— **Temps par bit** : $\frac{1}{1\,000\,000} = 1$ microseconde
— **Temps par octet** : 10 µs (10 bits × 1 µs)
— **Mode** : Half-duplex (une seule direction à la fois)
— **Niveau électrique** : TTL (0V = logique 0, 3.3V/5V = logique 1)

A.1.3 Protocole de communication de haut niveau (Feetech Protocol 0)

Les commandes sont envoyées sous forme de paquets structurés :

Structure d'un paquet (exemple : écriture de position)

```
[0xFF][0xFF]  : En-tête (2 octets)
[ID]          : Identifiant du moteur (1 octet, 1-6)
[LENGTH]      : Longueur du paquet (1 octet)
[INSTRUCTION] : Code d'instruction (1 octet)
                0x03 = READ (lecture)
                0x83 = SYNC_WRITE (écriture synchrone)
[ADDRESS]     : Adresse mémoire (1-2 octets)
                0x2A = Goal_Position (position cible)
                0x38 = Present_Position (position actuelle)
[DATA]        : Données (1-4 octets selon le paramètre)
[CHECKSUM]    : Somme de contrôle (1 octet)
```

A.1.4 Mécanisme d'adressage sur bus partagé

Tous les servomoteurs écoutent le même bus simultanément :

— **Commande individuelle** : Le paquet contient l'ID spécifique (ex : ID=3). Seul le moteur 3 répond et exécute.
— **Commande de diffusion (SYNC_WRITE)** : Le paquet contient ID=0xFE (broadcast) avec les données pour tous les moteurs. Chaque moteur extrait ses propres données selon son ID et exécute sans répondre (plus rapide).

Exemple de paquet SYNC_WRITE pour 3 moteurs :
`[0xFF][0xFF][0xFE][LENGTH][0x83][0x2A][0x02]`
`[ID=1][POS_LOW][POS_HIGH]`
`[ID=2][POS_LOW][POS_HIGH]`
`[ID=3][POS_LOW][POS_HIGH]`
`[CHECKSUM]`

A.1.5 Tableau de contrôle des servomoteurs

Chaque servomoteur possède une mémoire interne organisée en tableau de contrôle (registres), telle que détaillée au tableau A.1 :

TABLEAU A.1 – Tableau de contrôle des servomoteurs Feetech

Paramètre	**Adresse**	**Taille**	**Valeurs**
ID	5	1 octet	0-253
Baud_Rate	6	1 octet	0=1Mbps, 1=500kbps...
Goal_Position	42	2 octets	0-4095 (12 bits)
Present_Position	56	2 octets	0-4095 (lecture seule)
Present_Temperature	63	1 octet	Température en °C
Present_Current	69	2 octets	Courant en mA

A.1.6 Flux de communication détaillé

Étape 1 - Génération de commande (Python) :

```
bus.sync_write("Goal_Position", {
    "shoulder_pan": 2048,
    "shoulder_lift": 1500,
    ...
})
```

Étape 2 - Sérialisation : Le code Python convertit les positions en octets (little-endian : octet de poids faible d'abord)

Étape 3 - Transmission USB → UART : Le driver USB (`/dev/ttyACM0`) envoie les octets via PySerial

Étape 4 - Conversion USB-TTL : La carte Bus Servo Driver convertit les paquets USB en signaux UART TTL

Étape 5 - Bus série partagé : Le signal UART parcourt tous les moteurs via leurs connecteurs d'entrée/sortie

Étape 6 - Réception et filtrage : Chaque moteur :

— Détecte l'en-tête [0xFF][0xFF]

— Lit l'ID du paquet

— Si ID correspond OU ID=broadcast : traite le paquet

— Sinon : ignore le paquet

Étape 7 - Exécution : Le moteur écrit la valeur à l'adresse spécifiée dans son tableau de contrôle

Étape 8 - Réponse (optionnelle) : Pour les commandes non-broadcast, le moteur renvoie un paquet de statut via le même bus

A.2 Implémentation logicielle de l'agent

Cette section regroupe les implémentations détaillées des composants logiciels de l'agent d'orchestration LangGraph.

A.2.1 Configuration du graphe d'agent LangGraph

La figure A.2 présente l'implémentation complète du graphe d'exécution de l'agent. Ce graphe définit la structure de flux entre les nœuds de raisonnement, d'exécution d'outils et de normalisation, ainsi que la gestion de l'état conversationnel via le `MemorySaver`.

A.2.2 Configuration des modèles LLM

La figure A.3 détaille la configuration des deux modèles LLM utilisés dans l'architecture. Le modèle principal `gemini-2.5-flash` gère le raisonnement et la prise de décision, tandis que le modèle `gemini-2.5-flash-lite` est dédié à la normalisation des réponses avec une latence réduite.

A.2.3 Outils spécialisés de l'agent

Les listings suivants présentent les implémentations des outils (*tools*) mis à disposition de l'agent pour interagir avec son environnement.

```python
# Definition de l'etat du graphe
class State(TypedDict):
    messages: Annotated[list, add_messages]

def setup_agent():
    graph_builder = StateGraph(State)

    # Ajout des noeuds
    graph_builder.add_node("tools", ToolNode(TOOLS))
    graph_builder.add_node("chatbot", chatbot_node)
    graph_builder.add_node("normalization", normalization_node)

    # Definition des transitions
    graph_builder.add_edge("tools", "chatbot")

    # Logique conditionnelle pour la normalisation ou l'appel d'outils
    graph_builder.add_conditional_edges(
        "chatbot",
        should_normalize,
        {
            "tools": "tools",
            "normalization": "normalization"
        },
    )

    graph_builder.set_entry_point("chatbot")
    memory = MemorySaver()
    return graph_builder.compile(checkpointer=memory)
```

FIGURE A.2 – Configuration du graphe d'agent et définition de l'état

```python
chat_llm = ChatOpenAI(
    openai_api_key=os.getenv("OPENROUTER_API_KEY"),
    openai_api_base=os.getenv("OPENROUTER_BASE_URL"),
    model_name="google/gemini-2.5-flash",
    temperature=0,
    max_tokens=8096,
    max_retries=2,
    streaming=False,
)

normalization_llm = ChatOpenAI(
    openai_api_key=os.getenv("OPENROUTER_API_KEY"),
    openai_api_base=os.getenv("OPENROUTER_BASE_URL"),
    model_name="google/gemini-2.5-flash-lite",
    temperature=0,
    max_tokens=8096,
    max_retries=2,
    streaming=True,
)
```

FIGURE A.3 – Configuration des modèles LLM (principal et normalisation)

a) Outil de récupération de documentation technique

La figure A.4 implémente l'outil de récupération documentaire permettant à l'agent d'accéder aux spécifications techniques du robot et des algorithmes utilisés.

```python
@tool
def get_tech_doc(document_to_retrieve: str) -> str:
    """
    RECUPERATION DE DOCUMENTATION TECHNIQUE - Acces aux specifications detaillees
    Args:
        document_to_retrieve (str): Doit etre l'un des suivants :
        so101, real_time, ssl_yolo
    """
    file_path = os.path.join("docs", f"{document_to_retrieve}.md")
    try :
        with open(file_path, "r", encoding="utf-8") as file:
            content = file.read()
    except FileNotFoundError:
        return f"Document '{document_to_retrieve}' non trouve."
    return content
```

FIGURE A.4 – Outil de récupération de documentation technique

b) Outil de vision et analyse d'environnement

La figure A.5 présente l'implémentation de l'outil de vision exploitant les capacités multimodales du modèle Gemini Flash pour l'analyse de scène en temps réel.

c) Outil d'inspection de défauts YOLO

La figure A.6 détaille l'intégration du modèle SSL-YOLO pour la détection automatisée de défauts sur surfaces métalliques dans un contexte de contrôle qualité industriel.

A.2.4 Validation post-exécution

La figure A.7 illustre le mécanisme de validation en boucle fermée, où chaque action physique est suivie d'une vérification visuelle automatique pour confirmer l'atteinte de l'objectif.

```python
@tool
def capture_and_describe_image(prompt: str =
 "Decrivez le contenu de cette image.") -> str:
    """
    OUTIL D'EVALUATION DE L'ENVIRONNEMENT -
    Utilisez ceci pour comprendre l'etat de l'espace de travail.
    """
    # ... Initialisation de la camera et capture (code omis) ...

    # Encodage de l'image en base64
    image_base64 = base64.b64encode(buffer).decode('utf-8')

    # Appel au modele multimodal (Gemini Flash)
    message = HumanMessage(
        content=[
            {"type": "text", "text": prompt},
            {"type": "image_url",
             "image_url": {"url": f"{image_base64}"}},
        ]
    )
    response = description_llm.invoke([message])
    return response.content
```

FIGURE A.5 – Implémentation de la vision par ordinateur via VLM

```python
@tool
def detect_steel_defects() -> str:
    """
    OUTIL D'INSPECTION SPECIALISE - Detection de defauts sur l'acier
    par vision par ordinateur.
    """
    # ... Chargement des chemins (code omis) ...
    try:
        # Chargement du modele YOLO
        model = YOLO(str(model_path))

        # Inference sur l'image capturee
        results = model(str(image_path))

        # Traitement des resultats
        if len(results) > 0 and results[0].boxes is not None:
            boxes = results[0].boxes
            # ... Formatage du texte de resultat (classe, confiance) ...
            return result_text
        else:
            return "Aucun defaut detecte dans l'image."

    except Exception as e:
        return f"Erreur lors de la detection : {str(e)}"
```

FIGURE A.6 – Intégration du modèle YOLO pour l'inspection industrielle

```python
# Extrait de execute_robot_task
finally:
    # ... Deconnexion du robot ...
    if task_completed:
        # Validation automatique par vision
        validation_result = validate_task_with_camera(
            task_config.validation_message, camera_index=0
        )
        return f"Resultat de validation :\n{validation_result}"
```

FIGURE A.7 – Logique de validation post-exécution par vision

A.3 Modules de communication vocale

Cette section présente les implémentations des modules d'interface vocale permettant l'interaction naturelle entre l'opérateur et le système robotique.

A.3.1 Synthèse vocale (TTS)

La figure A.8 détaille l'implémentation du module de synthèse vocale utilisant le modèle `gemini-2.5-flash-preview-tts`. Le flux audio est diffusé en temps réel via PyAudio pour minimiser la latence perçue par l'utilisateur.

A.4 Résultats détaillés des évaluations sur 5 séries de 100 essais

Cette annexe présente les résultats des évaluations complémentaires menées sur la tâche de tri du bloc. Pour chaque configuration, 5 séries des 100 positions de test ont été évaluées, totalisant 500 essais par configuration.

Méthode statistique.

Les résultats de chaque configuration étant binaires (succès = 1, échec = 0), la significativité statistique des différences observées est évaluée à l'aide d'un modèle Linéaire Généralisé Binomial (GLM Binomial), équivalent à une régression logistique. Cette méthode est particulièrement adaptée pour comparer des taux de succès pour fournir des coefficients interprétables et permet des comparaisons multiples contrôlées. L'implémentation s'appuie sur la bibliothèque Python `statsmodels`, une solution statistique établie appliquée à nos données d'évaluation.

```python
def speak_text(text: str) -> None:
    response = client.models.generate_content(
        model="gemini-2.5-flash-preview-tts",
        contents=text,
        config=types.GenerateContentConfig(
            response_modalities=["AUDIO"],
            speech_config=types.SpeechConfig(
                voice_config=types.VoiceConfig(
                    prebuilt_voice_config=types.PrebuiltVoiceConfig(
                        voice_name='Kore',
                    )
                )
            ),
        )
    )
    # Lecture du flux audio via PyAudio
    data = response.candidates[0].content.parts[0].inline_data.data
    stream.write(data)
```

FIGURE A.8 – Génération de la synthèse vocale en streaming

```python
import statsmodels.formula.api as smf

glm_model = smf.glm(
    formula="success ~ C(model)", data=df, family=sm.families.Binomial()
)
glm_result = glm_model.fit()
```

FIGURE A.9 – Implémentation du modèle linéaire généralisé avec statsmodels

Pour chaque évaluation, le test du rapport de vraisemblance (LR, statistique χ^2 avec df degrés de liberté, où df = nombre de configurations -1) compare la configuration utilisé comme notre variable au modèle nul qui suppose un taux de succès identique pour toutes les configurations. Le pipeline de calcul est le suivant : le GLM estime un coefficient β pour chaque configuration, puis le ratio β/erreur-standard produit un z-score, dont on déduit la p-valeur ; en parallèle, e^{β} donne le rapport des cotes (OR). La p-valeur indique *si* une différence existe, tandis que l'OR quantifie *l'ampleur* de cette différence : OR > 1 signifie que la configuration comparée a de meilleures chances de succès que la référence, OR < 1 l'inverse. Pour les groupes de trois modèles, des comparaisons par paires sont effectuées avec une correction de **Bonferroni** (seuil ajusté $\alpha' = 0.05/k$, k = nombre de paires) pour limiter les faux positifs.

Avant de présenter les résultats détaillés, il convient de définir les catégories d'erreurs répertoriées lors des évaluations. Ces catégories constituent une taxonomie exhaustive des modes de défaillance possibles pour une tâche de *pick-and-place*, établie a priori pour couvrir l'ensemble des scénarios d'échec envisageables. Il est important de noter que toutes les catégories n'ont pas été observées lors de nos 3 000 essais cumulés (6 configurations $\times$ 500 essais). En pratique, seules les catégories C1, C3, C4, C5 et C8 ont été effectivement rencontrées, les catégories C2, C6 et C7 n'ayant jamais été observées dans nos expérimentations.

L'ensemble de ces erreurs trouve son origine dans les prédictions incorrectes des valeurs articulaires par les modèles VLA. Chaque catégorie représente une manifestation différente d'une même cause fondamentale : le modèle génère des positions articulaires qui dévient d'une trajectoire optimale, et cette déviation se traduit par des comportements erronés distincts selon la phase de la tâche et l'amplitude de l'erreur de prédiction.

Les catégories d'erreurs, incluant à la fois les échecs et les erreurs corrigées (récupérations) de manière autonome par les modèles, sont définies comme suit :

- **Catégorie 1 (C1)** : Erreur de localisation à la saisie. Le modèle prédit des valeurs articulaires qui positionnent le préhenseur à côté du bloc, résultant en une fermeture de la pince dans le vide. C'est la catégorie d'erreur la plus fréquente, représentant la majorité des échecs pour les trois architectures.
- **Catégorie 2 (C2)** : Collision de trajectoire avec l'environnement. Le modèle génère une séquence d'actions dont la trajectoire résultante entre en contact avec un obstacle de l'espace de travail.
- **Catégorie 3 (C3)** : Singularité cinématique. Le modèle prédit des configurations articulaires proches d'une singularité, provoquant un blocage temporaire.
- **Catégorie 4 (C4)** : Atteinte des limites de l'espace de travail. Bien que l'environnement soit contrôlé et que les positions prédéfinies du bloc et du conteneur se trouvent dans la portée physique du robot, quelques cibles approchent des limites de l'espace de travail. Le modèle peine à inférer les configurations articulaires optimales nécessaires pour atteindre ces positions extrêmes avec précision.
- **Catégorie 5 (C5)** : Déplacement du bloc hors de la zone de préhension. Cette erreur est une conséquence directe d'une erreur de localisation à la saisie (C1) : le préhenseur, mal positionné par rapport au bloc en raison de prédictions articulaires imprécises, entre en contact avec celui-ci lors de la

fermeture de la pince et le pousse hors de la zone accessible du bras, rendant toute tentative de récupération impossible.

— **Catégorie 6 (C6)** : Glissement de l'objet. Le modèle prédit une force de préhension insuffisante ou un angle d'approche inadapté, entraînant un glissement de l'objet lors du mouvement.

— **Catégorie 7 (C7)** : Relâchement prématuré en transport. Le modèle génère une commande d'ouverture de la pince avant d'atteindre la zone de dépôt.

— **Catégorie 8 (C8)** : Erreur de localisation au dépôt. Similaire à C1, mais survenant lors de la phase de placement : le modèle prédit des positions articulaires qui relâchent l'objet à côté du récipient cible plutôt qu'à l'intérieur.

A.4.1 Comparaison des architectures sur 5 séries de 100 essais

La figure A.10 présente la comparaison des trois architectures (GR00T, SmolVLA, Pi0.5) sur ces évaluations, avec une taille de lot de 64.

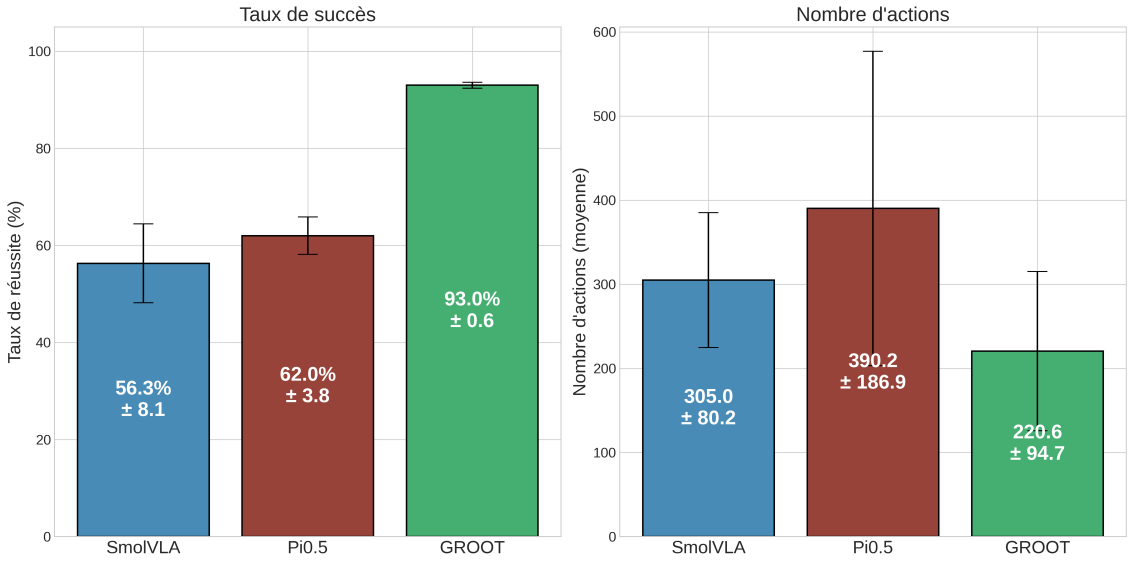


FIGURE A.10 – Comparaison des performances des trois architectures sur les 5 évaluations

Sur les 5 séries d'évaluation, GR00T maintient un taux de succès moyen de 93%, stable entre 92% et 94% selon la série. Pi0.5 obtient un taux moyen de 62%, avec une progression de 56% à 67%. SmolVLA atteint un taux moyen de 56.3%, variant de 44.6% à 66%. Ces résultats globaux confirment la hiérarchie observée sur la première série de 100 essais dans le chapitre 3.

GR00T se distingue aussi par son efficacité, avec un temps d'inférence moyen de 11.15 s et 221 actions par épisode en moyenne. Pi0.5 nécessite 390 actions et 19.71 s en moyenne, tandis que SmolVLA se situe entre les deux avec 305 actions et 16.38 s, comme synthétisé au tableau A.2.

Analyse statistique (GLM Binomial). Un test du rapport de vraisemblance confirme une différence globale très hautement significative entre les trois architectures ($\chi^2 = 218.72$, $df = 2$, $p < 0.001$, ***). Les coefficients du modèle placent GR00T comme référence (probabilité de succès = 93,0%) : Pi0.5 a des chances de succès 8.1× inférieures (OR = 0.12, $p < 0.001$) et SmolVLA 10.3× inférieures (OR = 0.10, $p < 0.001$). En revanche, la différence entre SmolVLA (56.3%) et Pi0.5 (62.0%) est non significative ($p = 0.066$, n.s.) après correction de Bonferroni ($\alpha' = 0.0167$), indiquant des performances statistiquement comparables entre ces deux architectures.

A.4.2 Tolérance aux erreurs et récupérations

La figure A.11 compare la capacité des trois modèles de base (taille de lot 64) à se corriger après une erreur.

GR00T présente un taux de récupération de 67.3%, nettement supérieur à ceux de SmolVLA (20.1%) et Pi0.5 (21.4%). Ce comportement de correction autonome, non présent dans les données d'entraînement, contribue directement au taux de succès élevé de GR00T. Le modèle parvient mieux à se repositionner après une erreur de saisie et à effectuer une nouvelle tentative. Les catégories d'erreurs les plus fréquentes sont les erreurs de localisation à la préhension (Catégorie 1 - C1) pour les trois modèles, suivies du déplacement du bloc hors zone (Catégorie 5 - C5) et du glissement de l'objet (Catégorie 10 - C10).

A.4.3 Impact de l'horizon de prédiction sur les 5 évaluations

La figure A.12 compare SmolVLA avec un horizon de prédiction de 16 et de 50 actions sur les 5 séries d'évaluations.

L'écart se confirme sur l'ensemble des évaluations, détaillées dans le tableau A.3. SmolVLA avec un horizon de 16 atteint un taux de succès moyen de 56.3%, contre 28.2% pour l'horizon de 50. La configuration avec un horizon étendu (*chunk size*) présente aussi une instabilité plus marquée, avec des taux variant de 23% à 34% selon les séries testées. Les arrêts automatiques diminuent fortement (de 56.4% à

TABLEAU A.2 – Statistiques détaillées par série pour SmolVLA, GR00T et Pi0.5 (taille de lot = 64)

SmolVLA					
Axe d'évaluation	**Série 0**	**Série 1**	**Série 2**	**Série 3**	**Série 4**
Taux de succès	44.6%	50.0%	66.0%	57.0%	64.0%
Temps d'inférence moyen (s)	16.36	16.58	16.25	16.11	16.59
Actions (moyenne ± écart-type)	297 ± 103.8	306 ± 142.7	310 ± 123.7	296 ± 92.8	317 ± 131.1
FPS	35.2	35.1	35.2	35.2	35.1
Arrêts automatiques	44.6%	52.0%	66.0%	55.0%	65.0%
Échecs (saisie / dépôt)	60 / 0	66 / 4	36 / 2	50 / 2	47 / 1
Catégories d'erreurs (Catégorie (échecs + récupérations))	C1 (46), C5 (12)	C1 (58), C5 (8), C8 (4)	C1 (30), C5 (6), C8 (2)	C1 (39), C5 (11), C8 (2)	C1 (39), C5 (8), C8 (1)
Total récupérations	4	22	6	10	12
GR00T					
Axe d'évaluation	**Série 0**	**Série 1**	**Série 2**	**Série 3**	**Série 4**
Taux de succès	92.0%	93.0%	93.0%	93.0%	94.0%
Temps d'inférence moyen (s)	11.71	11.15	11.06	11.17	10.64
Actions (moyenne ± écart-type)	235 ± 124.6	217 ± 104.1	223 ± 130.3	222 ± 93.1	206 ± 72.6
FPS	35.3	35.2	35.3	35.2	35.1
Arrêts automatiques	90.0%	90.0%	90.0%	92.0%	90.0%
Échecs (saisie / dépôt)	11 / 4	14 / 10	19 / 6	17 / 7	13 / 3
Catégories d'erreurs (Catégorie (échecs + récupérations))	C1 (11), C8 (4)	C1 (14), C8 (10)	C1 (18), C8 (6), C5 (1)	C1 (16), C8 (7), C5 (1)	C1 (11), C8 (3), C5 (2)
Total récupérations	8	17	18	17	10
Pi0.5					
Axe d'évaluation	**Série 0**	**Série 1**	**Série 2**	**Série 3**	**Série 4**
Taux de succès	56.0%	60.0%	62.0%	65.0%	67.0%
Temps d'inférence moyen (s)	18.81	19.09	21.00	20.11	19.55
Actions (moyenne ± écart-type)	370 ± 125.4	368 ± 123.2	416 ± 241.8	403 ± 153.0	393 ± 156.1
FPS	35.2	35.1	35.3	35.1	35.2
Arrêts automatiques	59.0%	62.0%	62.0%	64.0%	70.0%
Échecs (saisie / dépôt)	48 / 4	42 / 3	45 / 8	37 / 8	35 / 8
Catégories d'erreurs (Catégorie (échecs + récupérations))	C1 (41), C5 (6), C8 (4), C4 (1)	C1 (37), C8 (3), C5 (3), C4 (1)	C1 (37), C8 (8), C5 (8)	C1 (34), C8 (8), C5 (3)	C1 (29), C8 (8), C5 (5), C4 (1)
Total récupérations	9	5	15	11	11

29.4% en moyenne), indiquant que le modèle parvient moins souvent à terminer la tâche de manière autonome.

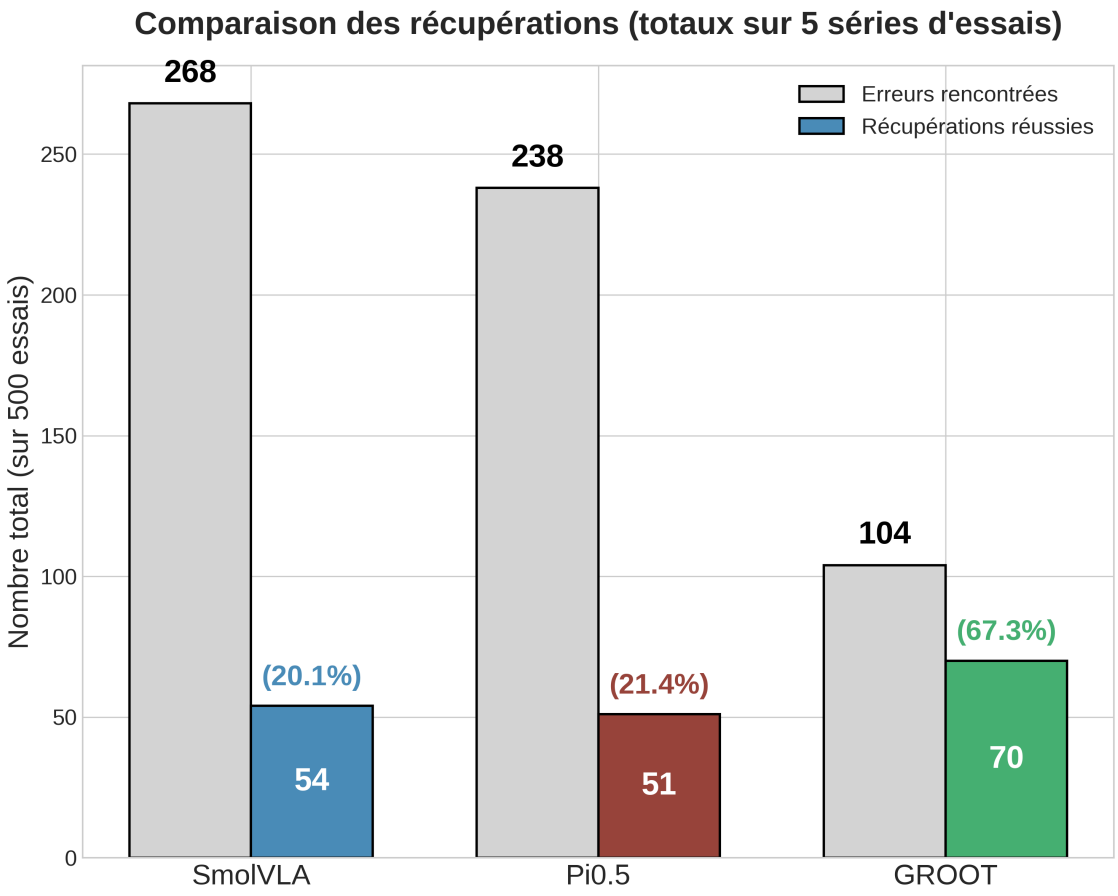


FIGURE A.11 – Comparaison des taux de récupération entre les trois architectures évaluées (5 évaluations)

TABLEAU A.3 – Statistiques détaillées par série pour SmolVLA (horizon = 50)

Axe d'évaluation	**Série 0**	**Série 1**	**Série 2**	**Série 3**	**Série 4**
Taux de succès	30.0%	28.0%	34.0%	26.0%	23.0%
Temps d'inférence moyen (s)	15.21	13.18	12.81	13.67	13.80
Actions (moyenne ± écart-type)	332 ± 136.8	287 ± 126.6	278 ± 123.0	295 ± 123.9	301 ± 125.1
FPS	35.2	35.2	35.2	35.2	35.2
Arrêts automatiques	32.0%	32.0%	30.0%	27.0%	26.0%
Échecs (saisie / dépôt)	74 / 6	67 / 9	68 / 4	76 / 4	80 / 5
Catégories d'erreurs (Catégorie (échecs + récupérations))	C1 (64), C5 (8), C8 (6), C4 (2)	C1 (57), C8 (9), C5 (8), C4 (2)	C1 (64), C8 (4), C4 (2), C5 (2)	C1 (65), C5 (10), C8 (4), C4 (1)	C1 (71), C5 (7), C8 (5), C4 (2)
Total récupérations	10	6	8	8	10

Analyse statistique (GLM Binomial). La différence entre les deux horizons est très hautement significative ($\chi^2 = 82.16$, $df = 1$, $p < 0.001$, ***). Un horizon de 50

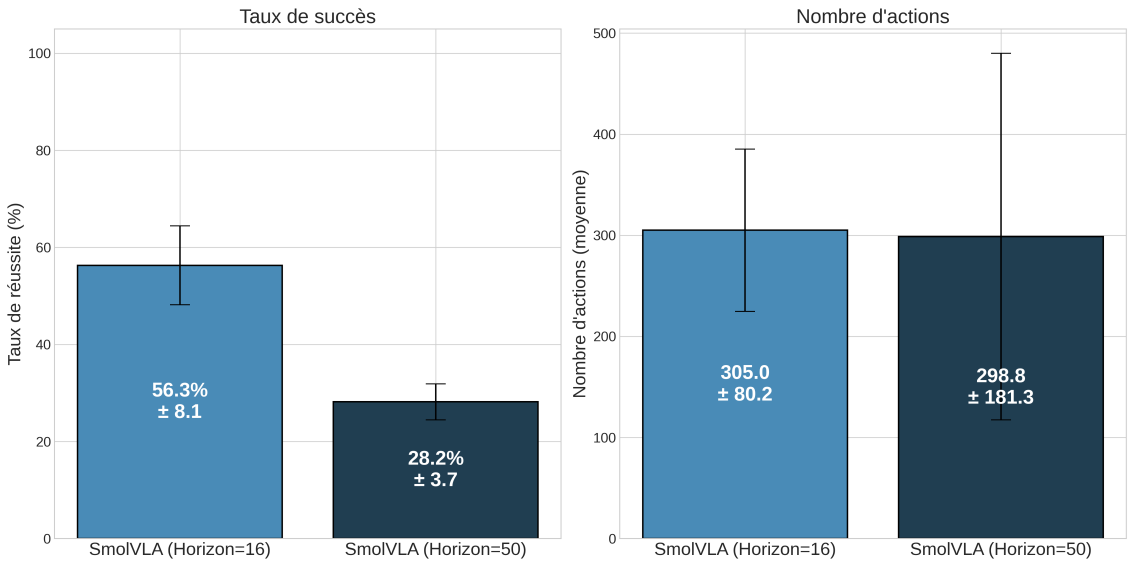


FIGURE A.12 – Impact de l'horizon de prédiction sur les performances de SmolVLA (5 évaluations)

actions multiplie par 3.3 le risque d'échec par rapport à l'horizon de 16 (OR = 0.31), confirmant que l'accumulation d'erreurs de prédiction sur un horizon étendu dégrade substantiellement les performances.

A.4.4 Impact de la taille de lot sur les 5 évaluations

La figure A.13 compare GR00T entraîné avec une taille de lot (*batch size*) de 64 et de 120, à travers les 5 séries de 100 essais.

Les multiples séries de test, détaillées dans le tableau A.4, confirment l'avantage d'une taille de lot plus grande. GR00T avec un lot de 120 atteint un taux de succès moyen de 98.6%, contre 93% avec un lot de 64. Le nombre d'échecs passe de 35 (lot de 64) à seulement 7 (lot de 120) sur la totalité des 500 essais cumulés. Le taux de récupération augmente également, passant de 70 à 75 récupérations totales.

Analyse statistique (GLM Binomial). Malgré un écart absolu de seulement 5.6 points, la supériorité de bs=120 est très hautement significative ($\chi^2 = 21.20$, $df = 1$, $p < 0.001$, ***). bs=64 présente des chances de succès 5.3× inférieures à bs=120 (OR = 0.19) : à ce niveau de performance, réduire les échecs de 35 à 7 sur 500 essais constitue un gain statistiquement robuste.

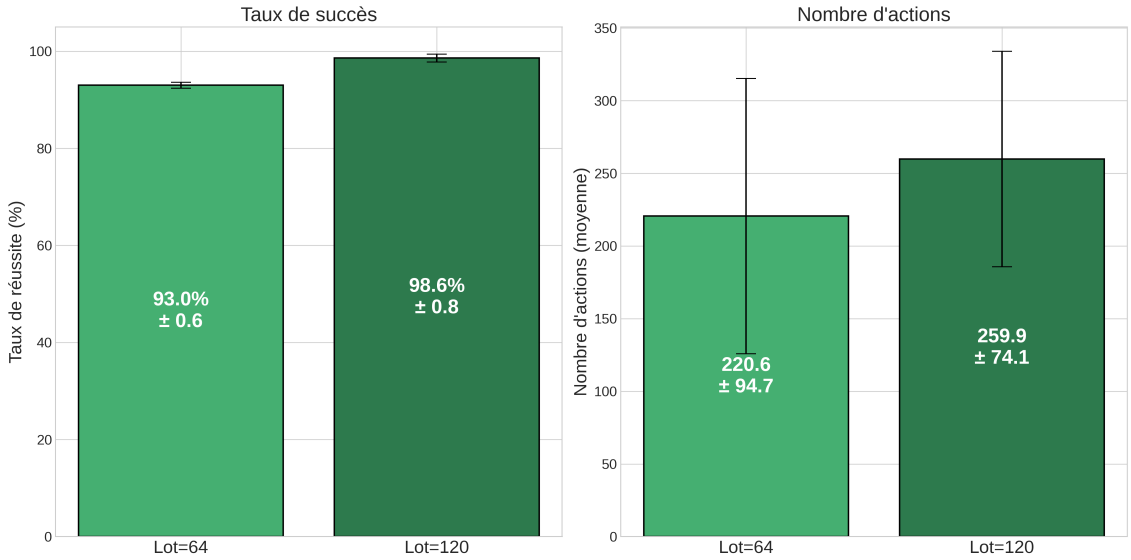


FIGURE A.13 – Impact de la taille de lot sur les performances de GR00T (5 évaluations)

TABLEAU A.4 – Statistiques détaillées par série pour GR00T (taille de lot = 120)

Axe d'évaluation	**Série 0**	**Série 1**	**Série 2**	**Série 3**	**Série 4**
Taux de succès	98.0%	100.0%	98.0%	98.0%	99.0%
Temps d'inférence moyen (s)	12.60	12.90	13.91	12.99	12.81
Actions (moyenne ± écart-type)	254 ± 150.2	258 ± 183.9	274 ± 175.1	257 ± 178.7	256 ± 172.7
FPS	35.2	35.2	35.2	35.2	35.2
Arrêts automatiques	100.0%	100.0%	98.0%	99.0%	100.0%
Échecs (saisie / dépôt)	10 / 4	12 / 8	12 / 8	10 / 3	10 / 5
Catégories d'erreurs (Catégorie (échecs + récupérations))	C1 (10), C8 (4)	C1 (12), C8 (8)	C1 (12), C8 (8)	C1 (10), C8 (3)	C1 (10), C8 (5)
Total récupérations	12	20	18	11	14

A.4.5 Impact de la distillation sur les 5 évaluations

La figure A.14 compare le modèle SmolVLA avant et après distillation (*knowledge distillation*) sur les 5 séries de 100 essais.

Le taux de succès moyen sur l'ensemble des séries passe de 56.3% à 81.6% après distillation, soit un gain de plus de 25 points de pourcentage, comme détaillé au tableau A.5. Le modèle distillé présente aussi une meilleure constance avec des taux variant de 72% à 88%, contre 44.6% à 66% pour le modèle initial. Le temps d'inférence moyen par épisode diminue également de 16.38 s à 13.65 s, et le nombre moyen d'actions passe de 305 à 229.

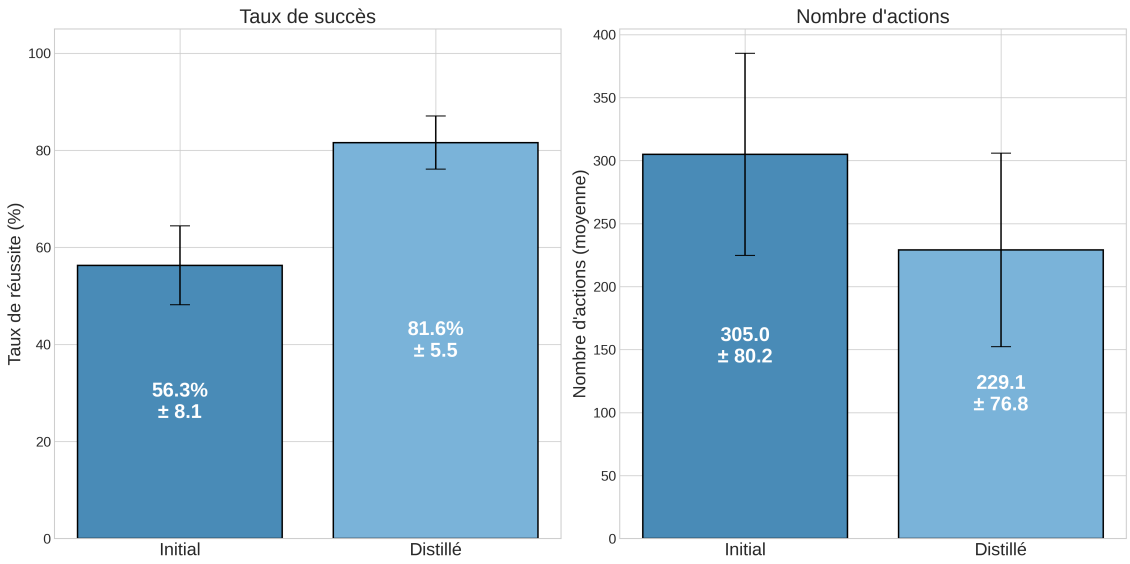


FIGURE A.14 – Impact de la distillation sur les performances de SmolVLA (5 évaluations)

TABLEAU A.5 – Statistiques détaillées par série pour SmolVLA distillé

Axe d'évaluation	**Série 0**	**Série 1**	**Série 2**	**Série 3**	**Série 4**
Taux de succès	80.0%	72.0%	88.0%	85.0%	83.0%
Temps d'inférence moyen (s)	13.37	14.20	13.59	13.57	13.52
Actions (moyenne ± écart-type)	239 ± 73.2	216 ± 56.9	231 ± 74.8	229 ± 70.3	231 ± 77.3
FPS	35.1	35.2	35.1	35.1	35.1
Arrêts automatiques	76.0%	78.0%	84.0%	85.0%	82.0%
Échecs (saisie / dépôt)	18 / 4	26 / 10	12 / 8	14 / 7	16 / 9
Catégories d'erreurs (Catégorie (échecs + récupérations))	C1 (14), C8 (4), C4 (2), C5 (2)	C1 (18), C8 (10), C5 (6), C4 (2)	C1 (10), C8 (8), C5 (2)	C1 (12), C8 (7), C5 (2)	C1 (11), C8 (9), C5 (3), C4 (2)
Total récupérations	2	10	8	7	9

Analyse statistique (GLM Binomial). La distillation produit un gain très hautement significatif ($\chi^2 = 76.55$, $df = 1$, $p < 0.001$, ***). Le modèle distillé multiplie par 3.44 les chances de succès par rapport au modèle original (OR = 3.44), confirmant l'efficacité de la distillation par connaissance.

Annexe B

Publications réalisées

1. R. Ghali, Z. Benhafid, and S. A. Selouani, "Real-time defect detection systems for steel and wood inspection," in *2024 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, Kingston, ON, Canada, 2024, pp. 577-582. DOI : 10.1109/CCECE59415.2024.10667164
2. R. Ghali, Z. Benhafid, and S. A. Selouani, "Benchmarking Few-Shot Learning Techniques for Steel Surface Defect Detection," in *2025 IEEE Smart World Congress (SWC)*, Calgary, AB, Canada, 2025, pp. 9–14. DOI : 10.1109/SWC65939.2025.00031
3. R. Ghali and S. A. Selouani, "ChatAcadien : A RAG-LLM-Based Chatbot for Exploring Acadian Genealogy," in *2025 IEEE International Conference on Collaborative Advances in Software and Computing (CASCON)*, Toronto, ON, Canada, 2025, pp. 62–67. DOI : 10.1109/CASCON66301.2025.00026
4. I. Kadri, S. A. Selouani, M. Ghribi, R. Ghali, and S. Mekhoukh, "LLM-driven agent for speech-enabled control of industrial robots : A case study in snow-crab quality inspection," *Results in Engineering*, vol. 27, p. 106660, 2025. DOI : https://doi.org/10.1016/j.rineng.2025.106660