



SORBONNE UNIVERSITÉ  
MASTER ANDROIDE

---

**Perception active pour la robotique  
basée sur des LLM**

---

UE de projet M1

Victor FLEISER – Thomas MARCHAND – Tarik Ege EKEN

2025

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>État de l'art</b>	<b>2</b>
<b>3</b>	<b>Contribution</b>	<b>3</b>
3.1	Prompting et modèles . . . . .	4
3.1.1	Exploration de prompts et modèles . . . . .	4
3.1.2	Affinage des prompts . . . . .	5
3.1.3	Robustesse aux conditions visuelles . . . . .	7
3.2	Incertitude . . . . .	11
3.2.1	Arbre de probabilités . . . . .	11
3.2.2	Exploration de suffixe . . . . .	14
3.2.3	Classification selon le suffixe . . . . .	15
3.3	Embeddings . . . . .	18
3.3.1	Principe et méthodologie . . . . .	18
3.3.2	Expérimentation et validation du concept . . . . .	19
3.3.3	Application à l'évaluation des réponses des VLMs . . . . .	19
3.3.4	Détermination du seuil optimal . . . . .	20
3.3.5	Analyse des cas limites . . . . .	21
3.4	Tests avec plusieurs Images . . . . .	23
3.5	Modèles de l'état de l'art . . . . .	25
<b>4</b>	<b>Conclusion</b>	<b>27</b>
<b>A</b>	<b>Cahier des charges</b>	<b>29</b>
<b>B</b>	<b>Manuel utilisateur</b>	<b>32</b>
B.1	Prompting et modèles . . . . .	32
B.2	Incertitude . . . . .	34
B.3	Embeddings . . . . .	36
<b>C</b>	<b>Exemples supplémentaires de résultats</b>	<b>38</b>
C.1	Prompting et Modèles . . . . .	38
C.2	Incertitude . . . . .	39
C.2.1	Arbre de probabilités . . . . .	39
C.2.2	Exploration de suffixe . . . . .	41
C.2.3	Classification selon le suffixe . . . . .	42

# Chapitre 1

## Introduction

Ce projet a été réalisé dans le cadre du parcours Androide de M1, sous la supervision de Stéphane Doncieux et Émiland Garrabé. Il s'inscrit dans une démarche exploratoire visant à intégrer les grands modèles de langage (LLM) dans des boucles de perception active pour la robotique, en particulier pour des tâches de classification d'objets par des robots manipulateurs tels que TIAGo ou des bras Franka.

Les grands modèles de language, notamment lorsqu'ils sont multimodaux (VLM : Vision-Language Model), ont montré un fort potentiel pour faciliter l'interaction entre des robots et des utilisateurs non-experts. Cependant, leur ancrage dans le monde réel demeure un défi, en particulier pour des versions légères adaptées à l'embarqué. Ces modèles peuvent produire des erreurs de reconnaissance ou manquer de confiance dans leurs décisions, notamment lorsque la qualité visuelle est dégradée ou que certains objets sont partiellement visibles. Cela rend pertinente l'approche de perception active, dans laquelle le robot interagit avec son environnement (par exemple en changeant de point de vue) pour améliorer sa compréhension.

L'objectif de ce projet est de concevoir et tester un module de perception active guidé par le language, capable d'exploiter des modèles VLM pré-entraînés dans un cadre interactif. A partir d'une image d'entrée montrant un objet tenu par un robot, le système tente de classifier l'objet à l'aide d'un VLM, puis décide s'il faut obtenir une nouvelle vue de l'objet pour lever l'incertitude. Plusieurs modèles et variantes de prompts ont été comparées pour identifier les stratégies les plus efficaces.

Le code et les ressources du projet sont disponibles sur le dépôt Github suivant :

<https://github.com/EGarrabe/Perception-active>

# Chapitre 2

## État de l'art

La perception active est un domaine clé de la robotique cognitive, qui vise à améliorer la compréhension d'une scène grâce à des interactions dynamiques. Contrairement à la perception passive, où l'agent traite des données sensorielles fixes, la perception active implique des actions ciblées comme déplacer la caméra ou manipuler l'objet pour obtenir de nouvelles informations visuelles. Cette capacité est particulièrement pertinente en robotique, où un agent peut repositionner son capteur ou un objet pour maximiser sa compréhension de la scène.

Avec l'émergence des grands modèles de langage multimodaux (VLMs), tels que LLaVA ou Gemma3, il est devenu possible d'envisager des stratégies de perception active combinant vision et raisonnement linguistique. Ces modèles peuvent analyser des images complexes et répondre à des questions ouvertes sur leur contenu, ce qui en fait des candidats naturels pour des scénarios interactifs où plusieurs vues sont exploitées pour améliorer la fiabilité de l'interprétation.

Cependant, la plupart des modèles les plus performants actuellement comme GPT-4o ou Gemini ne sont pas adaptés à un usage local ou embarqué. Leur exécution nécessite un accès à des serveurs distants, est coûteuse en ressources, et ne garantit pas le temps de réponse ni la confidentialité des données. À l'inverse, les modèles disponibles localement (via Ollama) sont plus limités en précision ou en capacités, mais permettent de concevoir des systèmes autonomes, exploitables en robotique réelle.

# Chapitre 3

## Contribution

Notre contribution s'inscrit dans une démarche expérimentale visant à évaluer la capacité de différents modèles VLM à classifier correctement des objets tenus par un bras robotique, et à déterminer dans quelles conditions la perception active permet d'améliorer les résultats.

Sélection des modèles et du jeu de test :

Nous avons testé plusieurs modèles open-source intégrés via Ollama en Python, incluant LLaVa, Gemma3, et LLaVa3.2-Vision, dans différentes tailles. Nous avons créé un jeu de données constitué de 60 images, représentant 12 objets différents tenus par un bras robot Franka, sous diverses orientations. Ces variations simulent les déplacements qu'un robot pourrait effectuer pour raffiner sa perception.

Les images ont été prises avec un téléphone, dans l'idée de reproduire approximativement le point de vue d'une caméra embarquée sur un robot manipulateur mobile tel que le TIAGo. Bien que le bras Franka ait été utilisé pour des raisons pratiques, l'objectif était d'obtenir un rendu visuel relativement proche de ce qu'une caméra de robot pourrait capter. Pour cela, les images ont été dégradées (résolution, compression, bruit, plage dynamique, balance des blancs, aliasing) afin de simuler la qualité typique d'une caméra embarquée :

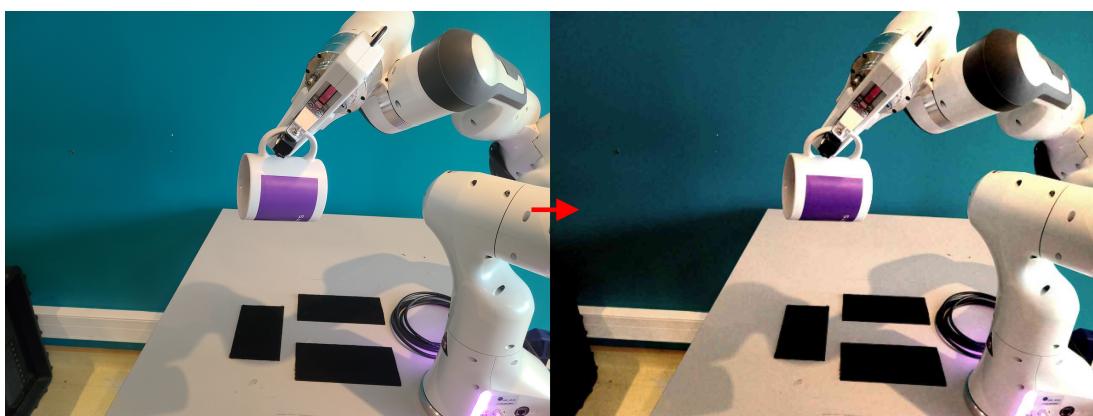


FIGURE 3.1 – Comparaison entre image d'origine et image manipulée

## 3.1 Prompting et modèles

Dans cette section nous avons cherché à comprendre ce qu'il faut inclure dans les prompts pour une réponse efficace pour la vision machine dans le contexte d'une image d'un bras robotique tenant un objet. L'autre aspect important de cette phase a été de voir l'impact que le choix du modèle a sur les résultats, c'est pourquoi chaque combinaison de prompt et d'image a également été testé sur chacun des 8 modèles suivants :

- llava :7b (7.24 milliards de paramètres)
- llava :13b (13 milliards de paramètres)
- llava :34b (34.4 milliards de paramètres)
- llava-llama3 (8.03 milliards de paramètres)
- llama3.2-vision (9.78 milliards de paramètres)
- gemma3 :4b (4.3 milliards de paramètres)
- gemma3 :12b (12.2 milliards de paramètres)
- gemma3 :27b (27.4 milliards de paramètres)

Afin d'assurer la reproductibilité des résultats, tous les appels ont été effectués avec une température fixée à 0.

### 3.1.1 Exploration de prompts et modèles

La première étape a consisté à tester un ensemble varié de prompts, dans le but d'identifier les formulations les plus efficaces et les plus robustes. Chaque prompt a été appliqué à la même série d'images sur l'ensemble des modèles sélectionnés.

	gemma3.12b	gemma3.27b	gemma3.4b	llama3.2-vision	llava	llava-llama3	llava.13b	llava.34b	
Act as a robotic assistant that identifies tools. What is in your grip?	31	30	29	40	25	22	28	20	225
Based on what you see, you must identify the object. Be concise, identify the object.	26	27	18	33	19	17	21	14	175
Based on what you see, you must identify the object. You will see a robotic arm in the image, but IGNORE IT, focus on the object. Be concise, identify the object.	28	30	24	35	25	21	25	25	213
Identify the item in the image	26	33	21	30	25	24	25	17	201
Ignore the robotic arm. Focus only on the object being held. Identify it.	29	33	23	39	25	26	26	22	223
What is the object in the image? Answer with a single word.	18	11	13	23	12	11	11	11	110
What is the object in the image? Ignore the robot arm, focus on the object. Answer with a single word.	22	22	23	32	17	22	20	23	181
You are a robot. Based on what you see, you must identify the object. Be concise, identify the object.	27	32	24	37	24	13	25	22	204
You are a robot. Based on what you see, you must identify the object. You will see a robotic arm in the image, but IGNORE IT, focus on the object. Be concise, identify the object.	29	31	27	36	27	22	25	26	223
You are a warehouse robot sorting objects. What are you holding?	34	35	28	38	17	26	23	21	222
You are analyzing this image to detect the item held. Disregard the arm or background clutter.	36	35	27	36	24	24	26	25	233
You must identify the object. You will see a robotic arm in the image, but IGNORE IT, focus on the object. Be concise, identify the object.	30	29	25	37	26	23	26	24	220
Your task is to classify the item. Do not describe the robot or background.	31	33	24	33	22	27	25	24	219
	367	361	306	449	288	278	306	274	TOTAL
Max	Max-1	Max-2	Max-3-5		Min+3-5	Min+2	Min+1	Min	RANK

FIGURE 3.2 – Résultats des 13 prompts d'exploration

#### Observation 1 – Tous les modèles ne se valent pas :

Parmi tous les modèles testés, `llama3.2-vision` s'est nettement démarqué avec les meilleurs scores pour presque tous les prompts. Les modèles `gemma3` arrivent ensuite, généralement devant la famille `llava`.

Une explication possible est que `llama3.2-vision` a été spécifiquement fine-tuné pour le raisonnement visuel, comme l'indique sa description officielle : "instruction-tuned image reasoning model". À l'inverse, les modèles `llava` et `gemma3` sont davantage conçus pour un usage multimodal plus général, sans ciblage aussi poussé sur des tâches d'analyse d'image.

## **Observation 2 – Un plus gros modèle n'est pas toujours un meilleur modèle :**

La taille d'un modèle ne garantit pas systématiquement de meilleures performances. Par exemple, dans le cas des modèles `llava`, les résultats ont suivi une tendance contre-intuitive : `llava:34b` (le plus grand) a obtenu les pires performances, tandis que `llava:13b` (modèle intermédiaire) a donné les meilleurs résultats, suivi de `llava:7b`.

À l'inverse, pour la famille `gemma3`, les performances ont augmenté de manière plus attendue avec la taille du modèle.

Cela montre qu'un modèle plus grand n'est pas nécessairement meilleur, et que l'augmentation du nombre de paramètres ne garantit pas à elle seule une amélioration des performances.

## **Observation 3 – Réponses longues vs Réponses courtes :**

Les prompts générant des réponses longues ont obtenu des scores légèrement meilleurs en moyenne, selon notre méthode d'évaluation (présence d'un terme correct dans la réponse). Toutefois, ce léger gain s'accompagne de plusieurs inconvénients notables :

- **Temps d'exécution rallongé** : les réponses longues prennent environ trois fois plus de temps à générer, ce qui constitue un coût important en contexte robotique temps réel.
- **Réponses moins exploitable**s : les réponses longues sont souvent trop descriptives et incluent des éléments non pertinents, comme des mentions du bras robotique, ce qui complique leur traitement automatique.

En raison de ces limitations, nous avons choisi de ne pas poursuivre l'exploration de prompts générant des réponses longues, et de privilégier les formulations visant des réponses concises et centrées sur l'identification de l'objet.

### **3.1.2 Affinage des prompts**

Après l'exploration initiale des prompts, cette section se concentre sur leur affinage, avec pour objectif d'augmenter la fiabilité des réponses générées. Nous étudions d'une part l'influence que peut avoir la formulation du contexte visuel dans le prompt, et d'autre part la construction de variantes ciblées à partir des premières formulations prometteuses. Ces expérimentations permettent de dégager des formulations plus robustes, qui serviront de base pour les évaluations ultérieures.

## **Observation 4 – Importance du contexte et de l'instruction explicite face au bras robotique :**

La présence du bras robotique dans l'image peut perturber l'interprétation du modèle, notamment lorsqu'il occupe une portion significative de l'image. En l'absence de consigne claire, le modèle peut tenter d'identifier le bras plutôt que l'objet manipulé, ou interpréter l'ensemble comme une structure unique.

Nous avons testé plusieurs variantes de prompts intégrant (ou non) deux éléments clés :

- Une instruction explicite d'**ignorer le bras robotique** : “*You will see a robotic arm in the image, but IGNORE IT, focus on the object.*”

- Un contexte d'usage cohérent où la présence d'un bras robotique est naturelle : “*You are a robot.*”

	gemma3:12b	gemma3:27b	gemma3:4b	llama3.2-vision	llava	llava-llama3	llava.13b	llava:34b	TOTAL
Based on what you see, you must identify the object. Be concise, identify the object.	26	27	18	33	19	17	21	14	175
Based on what you see, you must identify the object. You will see a robotic arm in the image, but IGNORE IT, focus on the object. Be concise, identify the object.	28	30	24	35	25	21	25	25	213
You are a robot. Based on what you see, you must identify the object. Be concise, identify the object.	27	32	24	37	24	13	25	22	204
You are a robot. Based on what you see, you must identify the object. You will see a robotic arm in the image, but IGNORE IT, focus on the object. Be concise, identify the object.	29	31	27	36	27	22	25	26	223

FIGURE 3.3 – Scores selon la présence ou absence de mentions liées au bras robotique ou le fait que il est un robot dans les prompts

Les résultats montrent clairement qu'ajouter l'un ou l'autre de ces éléments améliore significativement les performances. Le prompt ne contenant aucune de ces mentions obtient les plus mauvais scores, tandis que la combinaison des deux atteint les meilleurs. Cela suggère que ces indications aident le modèle à bien distinguer l'objet à identifier du reste de la scène.

À noter que pour les images sans bras robotique (images 48 à 60), l'ajout de ces mentions n'a pas d'effet négatif : le modèle semble capable d'ignorer l'instruction superflue sans dégradation des performances.

#### Recherche d'un prompt plus efficace :

À partir des résultats de l'exploration initiale (13 prompts variés), nous avons isolé les formulations et éléments de langage qui semblaient produire les meilleures performances. Sur cette base, nous avons construit 7 nouveaux prompts en combinant ces éléments jugés efficaces - notamment les formulations courtes, les instructions explicites, et les contextes adaptés à la présence du bras robotique.

	gemma3:12b	gemma3:27b	gemma3:4b	llama3.2-vision	llava	llava-llama3	llava.13b	llava:34b	
You are a robot. You are analyzing this image to detect the item held. You will see a robotic arm in the image, but IGNORE IT, focus on the object. Be concise, identify the object.	33	34	28	40	28	25	27	30	245
Act as a robotic assistant that identifies tools. Based on what you see, you must identify the object gripped in your robotic arm. Disregard the arm and background clutter, focus on the object. Be concise, identify the object.	32	34	25	40	29	27	26	28	241
You are analyzing this image to detect the item held. Based on what you see, you must identify the object gripped in your robotic arm. Disregard the arm and background clutter, focus on the object. Be concise, identify the object.	35	33	29	41	27	24	26	28	243
You are a robot. Based on what you see, you must identify the object gripped in your robotic arm. Disregard the arm and background clutter, focus on the object. Be concise, identify the object.	36	36	28	40	27	27	26	28	248
You are a robot. You are analyzing this image to detect the item held. Based on what you see, you must identify the object gripped in your robotic arm. Disregard the arm and background clutter, focus on the object. Be concise, identify the object.	34	33	27	41	27	24	26	26	238
You are a robot. You are analyzing this image to detect the item held. Disregard the arm and background clutter, focus on the object. Be concise, identify the object.	34	34	31	37	28	26	28	30	246
You are a robot. Based on what you see, you must identify the object gripped in your robotic arm. You will see a robotic arm in the image, but IGNORE IT, focus on the object. Be concise, identify the object.	35	34	26	42	28	25	27	28	245
	239	238	194	281	194	178	186	198	TOTAL
	2	3	5	1	5	8	7	4	RANK
Max	Max-1	Max-2	Max-3	Max-4	Max-5	Max-6	Max-7		

FIGURE 3.4 – Résultats des 7 prompts construits à partir des meilleures pratiques identifiées

Tous ces prompts présentent les scores les plus élevés obtenus jusqu'à présent, avec des performances globalement proches les unes des autres. Parmi eux, le prompt suivant a été retenu pour la suite des tests et des analyses du rapport :

“*You are a robot. Based on what you see, you must identify the object gripped in your robotic arm. Disregard the arm and background clutter, focus on the object. Be concise, identify the object.*”

## Résultats pour tous les prompts :

	gemma3.12b	gemma3.27b	gemma3.4b	llama3.2-vision	llava	llava-llama3	llava.13b	llava.34b	
	Max	Max-1	Max-2	Max-3	Max-3.5	Min+3-5	Min+2	Min+1	Min
Act as a robotic assistant that identifies tools. Based on what you see, you must identify the object gripped in your robotic arm. Disregard the arm and background clutter, focus on the object. Be concise, identify the object.	32	34	25	40	29	27	26	28	241 6
Act as a robotic assistant that identifies tools. What is in your grip?	31	30	29	40	25	22	28	20	225 9
Based on what you see, you must identify the object. Be concise, identify the object.	26	27	18	33	19	17	21	14	175 10
Based on what you see, you must identify the object. You will see a robotic arm in the image, but IGNORE IT, focus on the object. Be concise, identify the object.	28	30	24	35	25	21	25	25	213 15
Identify the item in the image	26	32	21	30	25	24	25	17	201 17
Ignore the robotic arm. Focus only on the object being held. Identify it.	29	33	23	39	25	26	22	223	11
What is the object in the image? Answer with a single word.	18	11	13	23	12	11	11	11	110 20
What is the object in the image? Ignore the robot arm, focus on the object. Answer with a single word.	22	22	23	32	17	22	20	23	181 18
You are a robot. Based on what you see, you must identify the object gripped in your robotic arm. Disregard the arm and background clutter, focus on the object. Be concise, identify the object.	36	36	28	40	27	27	26	28	248 1
You are a robot. Based on what you see, you must identify the object gripped in your robotic arm. You will see a robotic arm in the image, but IGNORE IT, focus on the object. Be concise, identify the object.	35	34	26	42	28	25	27	28	245 3
You are a robot. Based on what you see, you must identify the object. You will see a robotic arm in the image, but IGNORE IT, focus on the object. Be concise, identify the object.	27	32	24	37	24	13	25	22	204 16
You are a robot. Based on what you see, you must identify the object. You will see a robotic arm in the image, but IGNORE IT, focus on the object. Be concise, identify the object.	29	31	27	36	27	22	25	26	223 11
You are a robot. You are analyzing this image to detect the item held. Based on what you see, you must identify the object gripped in your robotic arm. Disregard the arm and background clutter, focus on the object. Be concise, identify the object.	34	33	27	41	27	24	26	26	238 7
You are a robot. You are analyzing this image to detect the item held. Disregard the arm and background clutter, focus on the object. Be concise, identify the object.	34	34	31	37	28	26	28	30	246 1
You are a robot. You are analyzing this image to detect the item held. You will see a robotic arm in the image, but IGNORE IT, focus on the object. Be concise, identify the object.	33	34	28	40	28	25	27	30	245 3
You are a warehouse robot sorting objects. What are you holding?	34	35	28	38	17	26	23	21	222 12
You are analyzing this image to detect the item held. Based on what you see, you must identify the object gripped in your robotic arm. Disregard the arm and background clutter, focus on the object. Be concise, identify the object.	35	33	29	41	27	24	26	28	243 5
You are analyzing this image to detect the item held. Disregard the arm or background clutter.	36	35	27	36	24	24	26	25	233 8
You must identify the object. You will see a robotic arm in the image, but IGNORE IT, focus on the object. Be concise, identify the object.	30	29	25	37	26	23	26	24	220 13
Your task is to classify the item. Do not describe the robot or background.	31	33	24	33	22	27	25	24	219 14
	606	619	500	730	482	456	492	472	TOTAL
	3	2	4	1	6	8	5	7	RANK

FIGURE 3.5 – Résultats des 20 prompts testés au total

### 3.1.3 Robustesse aux conditions visuelles

Pour évaluer la robustesse des modèles de vision à des conditions réalistes ou dégradées, nous avons soumis chaque image à 17 transformations simulant des perturbations possibles en environnement robotique : variations d'éclairage, de couleur, de qualité d'image, de géométrie ou de cadrage. Ces modifications permettent de simuler des situations courantes telles qu'une caméra de faible qualité, une mauvaise luminosité ou une mauvaise orientation de l'objet. Une illustration de ces transformations est présentée en début de cette section.

Nous avons ensuite évalué les performances des différents modèles sur l'ensemble de ces images modifiées afin d'observer dans quelle mesure chaque modèle est affecté par les différentes conditions, ainsi que les effets généraux de chacune de ses conditions. Cette évaluation met en évidence non seulement la robustesse des modèles, mais aussi leur capacité à exploiter des conditions favorables avec les 2 premières conditions qui constituent des améliorations de l'image.

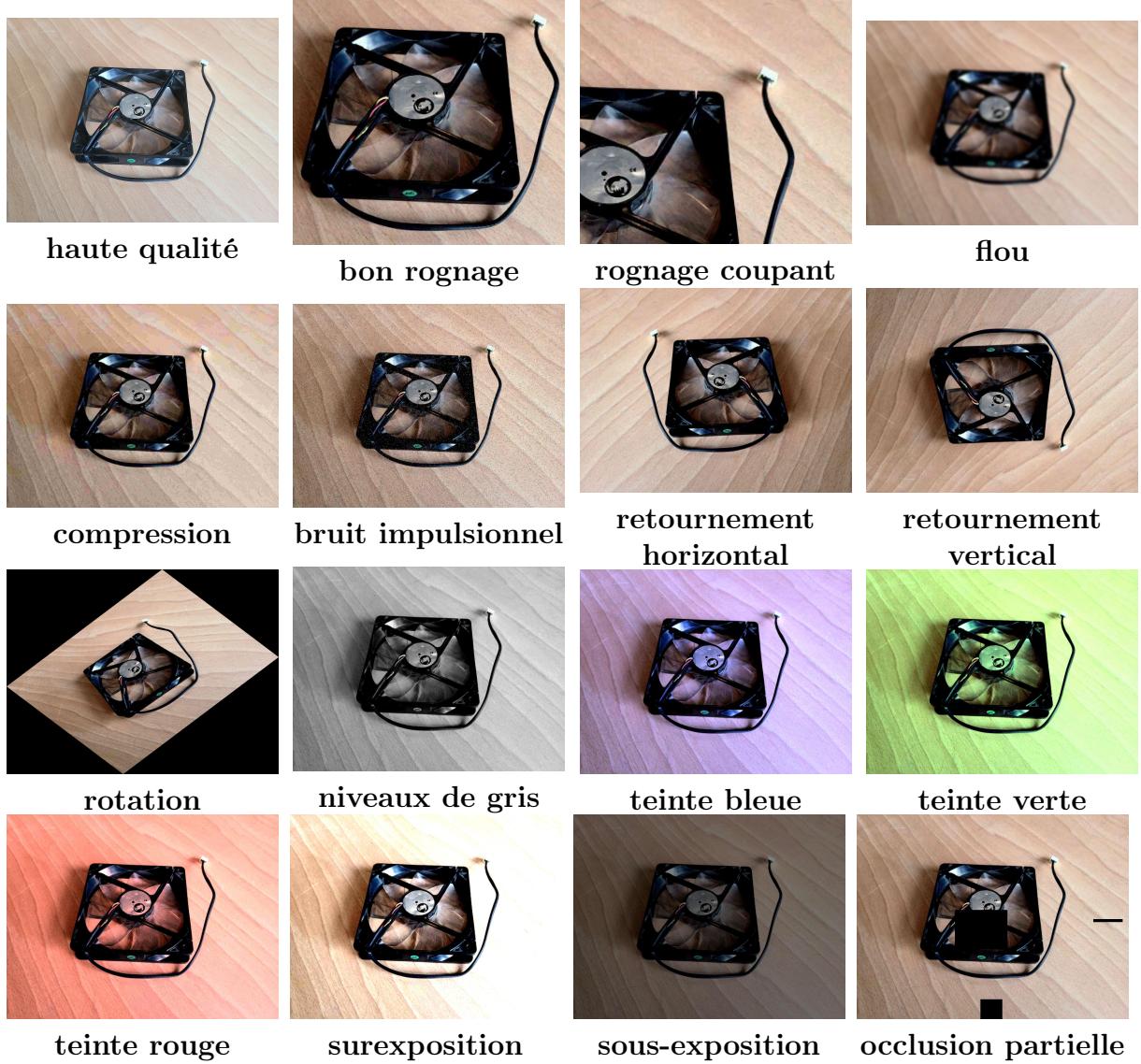


FIGURE 3.6 – Exemples des 16 conditions visuelles appliquées. (Certaines conditions sont difficiles à remarquer sur le rapport car les images sont rapetissées)

## Effet des différentes conditions visuelles :

	gemma3:12b	gemma3:27b	gemma3:4b	llama3.2-vision	llava	llava-llama3	llava:13b	llava:34b	TOTAL
blurred	-14	-19	-13	-10	-12	-9	-4	-3	-84
cut_object_crop	-10	-7	-15	-12	7	-7	7	8	-29
flipped_horizontal	-2	-4	-3	-2	-3	-5	2	0	-17
flipped_vertical	-9	-7	-4	1	-6	-11	-4	-2	-42
good_crop	1	-3	5	-2	9	4	12	10	36
grayscale	-2	-2	-4	-1	-8	-5	-4	-7	-33
high_quality	4	4	2	9	-1	1	9	4	32
images_base	0	0	0	0	0	0	0	0	0
jpeg_artifacts	-7	-9	-7	-2	-4	-2	2	2	-27
occluded	-6	-3	-4	-3	-7	-4	-1	-2	-30
overexposed	-7	-12	-1	-1	-3	-4	0	1	-27
rotated	-8	-8	-10	-1	-13	-11	-6	-3	-60
salt_pepper_noise	-12	-11	-9	-7	-13	-3	-1	-1	-57
tint_blue	-1	-6	-4	1	0	-3	-2	0	-15
tint_green	-4	-3	-4	1	-2	0	-2	2	-12
tint_red	-4	-5	-5	-1	1	-3	2	-1	-16
underexposed	-5	-3	-4	-1	-2	0	1	-1	-15
TOTAL	-86	-98	-80	-31	-57	-62	11	7	TOTAL

10 or more
7 to 9
4 to 6
1 to 3
0 -1 to -3
-4 to -6
-7 to -9
-10 or less

FIGURE 3.7 – Différences entre les résultats des images soumises aux différentes conditions et les résultats des images de base

Certaines transformations affectent peu les performances, notamment celles liées à la **couleur**. La conversion en niveaux de gris, les changements de teinte (rouge, verte, bleue), ainsi que les modifications d'exposition (surexposition et sous-exposition) provoquent une légère baisse des scores, mais restent globalement bien tolérées par l'ensemble des modèles.

Comme attendu, la **qualité d'image** joue un rôle significatif. Les images d'origine, capturées avec un appareil photo de smartphone mais volontairement dégradées pour simuler un flux vidéo de robot, obtiennent de moins bons résultats que les images haute qualité. Cela montre que la qualité de la caméra reste un facteur important dans l'identification d'objets.

Les **dégradations liées au bruit** (flou, bruit impulsionnel, compression JPEG) impactent plus fortement les performances, mais cet effet varie selon les modèles. Les modèles **llava** de grande taille (13b, 34b) montrent une bonne robustesse à ces perturbations, tandis que les modèles **gemma3** sont nettement plus affectés.

Concernant les **transformations géométriques**, le retournement horizontal a peu d'effet, mais le retournement vertical entraîne une dégradation notable, probablement parce que ce type de vue est peu fréquent dans les données d'entraînement des modèles. La rotation 45° entraîne une baisse très marquée des performances, cependant, cette chute peut être partiellement attribuée à des artefacts introduits lors du traitement (bords noirs, perte de qualité), plus qu'à la rotation elle-même.

Enfin, les **modifications de cadrage** révèlent des effets intéressants. Un rognage bien centré sur l'objet améliore généralement les performances, en particulier pour les modèles **llava**. Ce type de cadrage peut être obtenu automatiquement à l'aide de modèles de segmentation, ce qui est particulièrement pertinent dans un contexte robotique comme le nôtre. À l'inverse, un recadrage mal centré (où l'objet est partiellement visible et décalé)

dégrade les performances des modèles `gemma3` et `llama3.2-vision`, mais améliore paradoxalement celles des modèles `llava`, probablement en raison d'un effet de zoom implicite sur l'objet, malgré les défauts de cadrage.

### Comparaison de la robustesse entre modèles :

D'une manière générale, les modèles `llava`, en particulier les versions de plus grande taille (13b et 34b), apparaissent plus robustes face aux dégradations visuelles et savent mieux exploiter les transformations positives que les autres familles de modèles.

Cependant, il est important de noter que les modèles `gemma3` conservent des meilleurs résultats en général que les modèles `llava`, même sur les images modifiées. Cependant l'écart est plus faible grâce à la meilleure adaptabilité des modèles `llava`.

	gemma3:12b	gemma3:27b	gemma3:4b	llama3.2-vision	llava	llava-llama3	llava:13b	llava:34b	TOTAL
blurred	22	17	15	30	15	18	22	24	163
cut_object_crop	26	29	13	28	34	20	33	35	218
flipped_horizontal	34	32	25	38	24	22	28	27	230
flipped_vertical	27	29	24	41	21	16	22	25	205
good_crop	37	33	33	38	36	31	38	37	283
grayscale	34	34	24	39	19	22	22	20	214
high_quality	40	40	30	49	26	28	35	31	279
images_base	36	36	28	40	27	27	26	27	247
jpeg_artifacts	29	27	21	38	23	25	28	29	220
occluded	30	33	24	37	20	23	25	25	217
overexposed	29	24	27	39	24	23	26	28	220
rotated	28	28	18	39	14	16	20	24	187
salt_pepper_noise	24	25	19	33	14	24	25	26	190
tint_blue	35	30	24	41	27	24	24	27	232
tint_green	32	33	24	41	25	27	24	29	235
tint_red	32	31	23	39	28	24	28	26	231
underexposed	31	33	24	39	25	27	27	26	232
TOTAL	526	514	396	649	402	397	453	466	TOTAL

Max	Max-2	Max-4	Max-6	Min+6	Min+4	Min+2	Min

FIGURE 3.8 – Résultats des 17 conditions sur les modèles

## 3.2 Incertitude

Dans cette section, nous étudions plusieurs moyen d'utiliser les probabilités des tokens pour obtenir une mesure d'incertitude sur la réponse du VLM. Elle parle de la stratégie initiale de créer un arbre de probabilités, avant de revenir sur des méthodes plus simples mais plus utiles en pratique.

Tous les prompts utilisateur dans cette section vont utiliser le prompt optimal trouvé dans la section précédente **Prompting et modèles**.

### 3.2.1 Arbre de probabilités

Le but est d'obtenir une incertitude sur la réponse du VLM, pour savoir si on peut lui faire confiance. Notre idée initiale était de créer un arbre de probabilités qui couvre une grande partie des chemins possibles. Par exemple, quand on demande "What is the object ?" On peut avoir "The object is a mouse." avec une probabilité de 7%, "A computer mouse." avec une probabilité de 3%, etc...

La probabilité d'un chemin (séquence de tokens)  $S = (t_1, t_2, \dots, t_L)$  est calculé par le produit des probabilités conditionnelles des tokens à chaque étape :

$$P(S) = P(t_1) \times P(t_2|t_1) \times \cdots \times P(t_L|t_1, \dots, t_{L-1}) = \prod_{i=1}^L P(t_i|t_{<i})$$

Par exemple, pour "Computer mouse" avec les tokens  $t_1 = \text{"Computer"}$  et  $t_2 = \text{" mouse"}$ , la probabilité est  $P(t_1) \times P(t_2|t_1) = 0.54 \times 0.96 \approx 52\%$ .

Le nombre de réponses possibles croit exponentiellement avec la longueur de la réponse :

$$\text{total\_sequences} = \text{vocab\_size}^{\text{sequence\_length}}$$

où :

- **total\_sequences** est le nombre total de phrases possibles générées par le modèle,
- **vocab\_size** est la taille du vocabulaire disponible, typiquement 32000 pour LlaVA.
- **sequence\_length** est la longueur de la séquence générée en token.

(Cette formule est une approximation ; elle ne prend pas en compte les chemins qui terminent plus tôt, top-p<sup>1</sup> et top-k<sup>2</sup>, etc.)

En théorie, si nous testons tous les chemins possibles, nous pouvons analyser les chemins et voir dans quels cas le modèle identifie un objet correctement. Cependant, l'exploration exhaustive est impossible en pratique ; nous allons donc explorer l'arbre de manière selective.

Nous utilisons **llama-cpp-python**, qui donne accès à des méthodes permettant d'obtenir et modifier les probabilités des tokens suivants à chaque étape de la génération. Voici l'algorithme pour explorer l'arbre :

---

1. Le *top-p sampling* (ou *nucleus sampling*) consiste à sélectionner le plus petit ensemble de tokens dont la probabilité cumulée dépasse un certain seuil  $p$ . Le prochain token est ensuite échantillonné parmi cet ensemble réduit, permettant une diversité adaptative.

2. Le *top-k sampling* consiste à ne considérer que les  $k$  tokens les plus probables pour le choix du prochain token, puis à échantillonner parmi ceux-ci, limitant ainsi l'espace de recherche.

1. Génération du chemin le plus probable (rang 1 à chaque étape avec température 0).
2. À partir du deuxième chemin, on identifie des points de déviation par rapport aux chemins déjà explorés. À chaque étape d'un chemin, on considère les  $k$  tokens alternatifs les plus probables qui n'ont pas encore été explorés ou élagués.
3. Si un token alternatif mène immédiatement à un token de fin de séquence (EOS) ou à un mot cible, cette branche est "élaguée" : sa probabilité est calculée et ajoutée aux cumuls respectifs.
4. Sinon, le chemin est poursuivi en sélectionnant le token alternatif puis les plus probables jusqu'à atteindre la longueur maximale, un EOS, ou un mot cible.
5. Le code priorise l'exploration des chemins les plus probables.

Voici à quoi ressemble un arbre après 3 chemins :

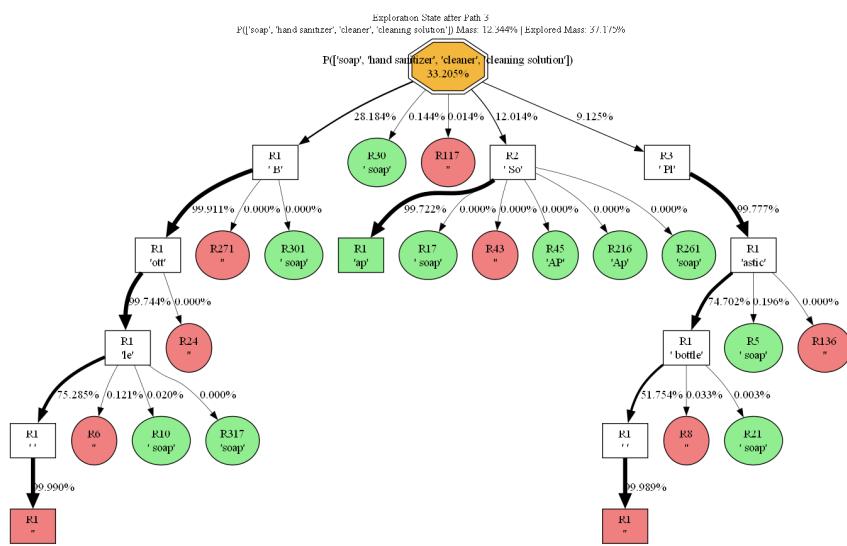


FIGURE 3.9 – Arbre de probabilités, avec 3 chemins (+ élagage) explorés.

Nous constatons que des mots peuvent être divisés en plusieurs tokens, et que plusieurs chemins sont élagués.

Cette méthode nous permet de construire progressivement un ensemble de chemins explorés,  $\mathcal{S}_{explored}$  (chemins complets générés et de branches élaguées), et un sous-ensemble  $\mathcal{V}_{explored} \subseteq \mathcal{S}_{explored}$  contenant les chemins considérés comme valides.

Un chemin est considéré comme valide s'il se termine par un mot cible parmi une liste de "bonnes réponses" prédéfinies. Cette méthode n'est pas parfaite car il faut définir les mots corrects, et nous supposons également que la réponse contient seulement la bonne réponse, sans avoir "not mouse" ou plusieurs objets. Dans les tests effectués ici, cette simplification n'a pas posé de problème.

La probabilité estimée d'obtenir une réponse correcte conditionnée à l'espace exploré, est alors calculée comme suit :

$$P(\text{correct} | \text{explored}) \approx \frac{\sum_{S \in \mathcal{V}_{explored}} P(S)}{\sum_{S' \in \mathcal{S}_{explored}} P(S')}$$

$\sum_{S \in \mathcal{V}_{explored}} P(S)$  : somme des probabilités de tous les chemins & branches valides trouvés  
 $\sum_{S' \in \mathcal{S}_{explored}} P(S')$  : somme des probabilités de tous les chemins & branches qui ont été complètement explorés ou élagués

Trouvons d'abord une bonne valeur pour le nombre de chemins à générer. Voici l'évolution des résultats sur un arbre contenant 500 chemins :

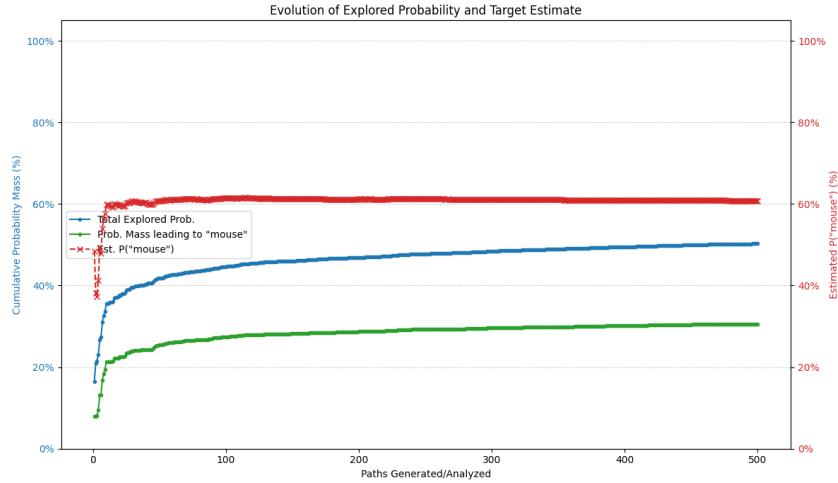


FIGURE 3.10 – Évolution de la probabilité d'avoir le mot correct, ainsi que les possibilités explorées

Nous observons que même après 500 chemins, seulement 50% de toutes les possibilités sont découverts (ligne bleue). La ligne rouge indique que la probabilité estimée d'avoir le mot correct parmi les réponses converge assez rapidement, aux alentours de 50 chemins.

Voici des résultats sur deux modèles, LlaVA 7B et LlaVA 13B, sur 4 images :

Image	Image Name	Target Word	---		LlaVA 7B		---		LlaVA 13B		---	
			Success	Explored	Time	Success	Explored	Time	Success	Explored	Time	Time
	49.jpg	'mug', 'cup'	99.292%	92.028%	987.4s	99.552%	89.570%	1553.2s				
	54.jpg	'soap', 'hand sanitizer', 'cleaner', 'cleaning solution'	36.549%	68.532%	892.0s	51.252%	67.001%	1578.1s				
	55.jpg	'cloth', 'rag', 'napkin'	42.125%	41.132%	886.4s	58.912%	48.426%	1532.0s				
	59.jpg	'cloth', 'rag', 'napkin'	24.434%	30.132%	875.1s	51.521%	51.377%	1757.7s				

FIGURE 3.11 – Probabilités pour 4 images différentes, calculées avec 50 chemins chacun.

Généralement, le plus grand modèle obtient de meilleurs résultats. Nous constatons également que le VLM a de meilleures performances pour les objets posés sur table.

Cette stratégie est intéressante car elle permet d'obtenir des probabilités pour des images dont le modèle arrive rarement à identifier.

Cependant, elle reste très intensive en ressources et n'est pas efficace pour déterminer l'incertitude des VLM. De plus, nous avons observé une manque de consistance : les objets représentables en un seul tokens sont élagués plus facilement et obtiennent une performance plus élevée.

C'est également ici que nous avons découvert une corrélation entre la performance et l'exploration au bout de 50 chemins : plus le VLM identifiait correctement l'objet, plus la totalité d'exploration était élevée. Nous avons donc décidé de ce concentrer sur les probabilités des tokens générés.

### 3.2.2 Exploration de suffixe

Puisque l'arbre n'est pas efficace, nous avons décidé de simplifier le procès, en générant non pas plus d'une vingtaine de réponse par image, mais seulement 3 réponses. Cette partie ne nécessite plus de mots "réponse", car nous regardons seulement les probabilités des tokens générés.

De plus, avec ce qu'on a appris, nous savons maintenant comment forcer le processeur VLM à générer un préfixe particulier : ici "The object is a". Cela nous permet d'avoir des réponses relativement courtes, rapides à générer et facilement comparables. Le but ici est de trouver une corrélation entre les objets identifiés correctement et les probabilités des tokens générés. Puisque la longueur de la réponse est variable, nous regardons ces valeurs pour chaque réponse dans le suffixe généré :

- Probabilité du premier token
- Probabilité du chemin complet (Produit des probabilités)
- Moyenne des probabilités

Voici quelques résultats :

Prompt: You are a robot. Based on what you see, you must identify the object gripped in your robotic arm. Disregard the arm and background clutter, focus on the object. Be concise, identify the object.  
Prefix: The object is a

Image	Image Name	LlaVA 7B (59.2s avg)				
		Suffix	1TkProb	Prob	AvgProb	Ranks
	04.jpg	✓ cup.	28.54%	24.82%	49.82%	(1, 1)
		✓ coffee cup.	19.61%	12.91%	50.54%	(2, 1, 1)
		✓ mug.	11.20%	10.17%	46.68%	(3, 1, 1)
	06.jpg	✗ piece of toast.	27.89%	7.93%	53.06%	(1, 1, 1, 1)
		✗ slice of pizza.	22.70%	13.45%	60.56%	(2, 1, 1, 1)
		✗ cookie.	7.99%	6.79%	26.05%	(3, 1)
	12.jpg	✓ glove.	41.14%	37.56%	72.15%	(1, 1, 1)
		✓ black and white glove.	18.60%	7.65%	65.15%	(2, 1, 1, 1, 1, 1)
		✓ pair of gloves.	7.66%	5.57%	48.59%	(3, 1, 1, 1)
	19.jpg	✓ fan.	33.24%	30.35%	55.09%	(1, 1)
		✗ camera.	7.43%	6.27%	25.03%	(2, 1)
		✓ small fan.	5.63%	2.02%	27.22%	(3, 1, 1)

FIGURE 3.12 – 12 réponses avec des statistiques sur les probabilités (4 images)

Il y a effectivement des corrélations entre une bonne identification des objets et les probabilités des tokens, mais il y a des cas spéciaux et il est difficile d'identifier un bon critère pour savoir si une réponse est correcte ou non. Un autre problème est l'identification d'un même objet mais d'une manière différente. Par exemple, "cup", "coffee cup" et "mug" sont tous corrects, mais nous l'évaluons pas ensemble.

**Pour la suite** Il peut être intéressant de trouver une méthode pour combiner les réponses correspondant à des objets identiques. Cela peut se faire par l'utilisation d'un LLM en plus qui combine les réponses similaires simplement. Une autre méthode est d'utiliser des embeddings pour comparer les mots similaires (Les embeddings sont explorés dans la section "Embeddings").

Ici, pour essayer de trouver un bon critère de séparation entre les objets bien reconnus et mals reconnus, nous avons décidé d'entraîner une classifieur sur les données acquises.

### 3.2.3 Classification selon le suffixe

Nous avons décidé d'entraîner un classifieur pour avoir une mesure d'incertitude sur la réponse du VLM.

Les 60 images ont été augmentées en effectuant des flip horizontaux et des rotations de 15 degrés. Nous utilisons donc les probabilités des 240 réponses sur les 240 images.

Chaque réponse (suffixe)  $S_{suffix} = (s_1, \dots, s_k)$  a une séquence de probabilités conditionnelles  $ProbS = (p_1, \dots, p_k)$ . La taille  $k$  de cette liste varie : il faut donc convertir cette liste en un vecteur de features de taille fixe.

Voici les features  $F = (f_1, f_2, \dots, f_7)$  utilisés pour chaque réponse, qui ont donné une bonne performance :

- Moyenne des probabilités :  $f_1 = \mu_p$
- Probabilité minimum :  $f_2 = \min(p_1, \dots, p_k)$
- Probabilité maximum :  $f_3 = \max(p_1, \dots, p_k)$
- Écart-type des probabilités :  $f_4 = \sigma_p$
- Probabilité du premier token :  $f_5 = p_1$
- Probabilité du dernier token :  $f_6 = p_k$
- Longueur de la réponse en token :  $f_7 = k$

Les labels  $y \in \{0, 1\}$  seront les réponses correctes :

Une réponse est considérée correcte si elle contient un mot parmi une liste de mots "corrects". Il faudrait potentiellement une meilleure méthode pour vérifier si une classification est correcte, mais vu le nombre limité de données, nous avons pu vérifier que ces labels sont corrects.

Nous utilisons une forêt d'arbres décisionnels (RandomForestClassifier) pour classifier les réponses.

Ce classifieur  $C$  apprend une fonction  $C : F \mapsto \hat{y}$ , où  $\hat{y} \in [0, 1]$  est la probabilité estimée que la réponse soit correcte. Cette valeur  $\hat{y}$  correspond à la confiance de la classification. La prédiction binaire est  $y_{pred} = 1$  si  $\hat{y} \geq \tau$ , et  $y_{pred} = 0$  sinon.

Le code teste plusieurs valeurs de seuil  $\tau \in [0, 1]$  pour trouver deux seuils optimaux : un seuil qui maximise accuracy, et un autre qui assure 0 Faux Positifs (FP) tout en maximisant accuracy. Voici les seuils trouvés :

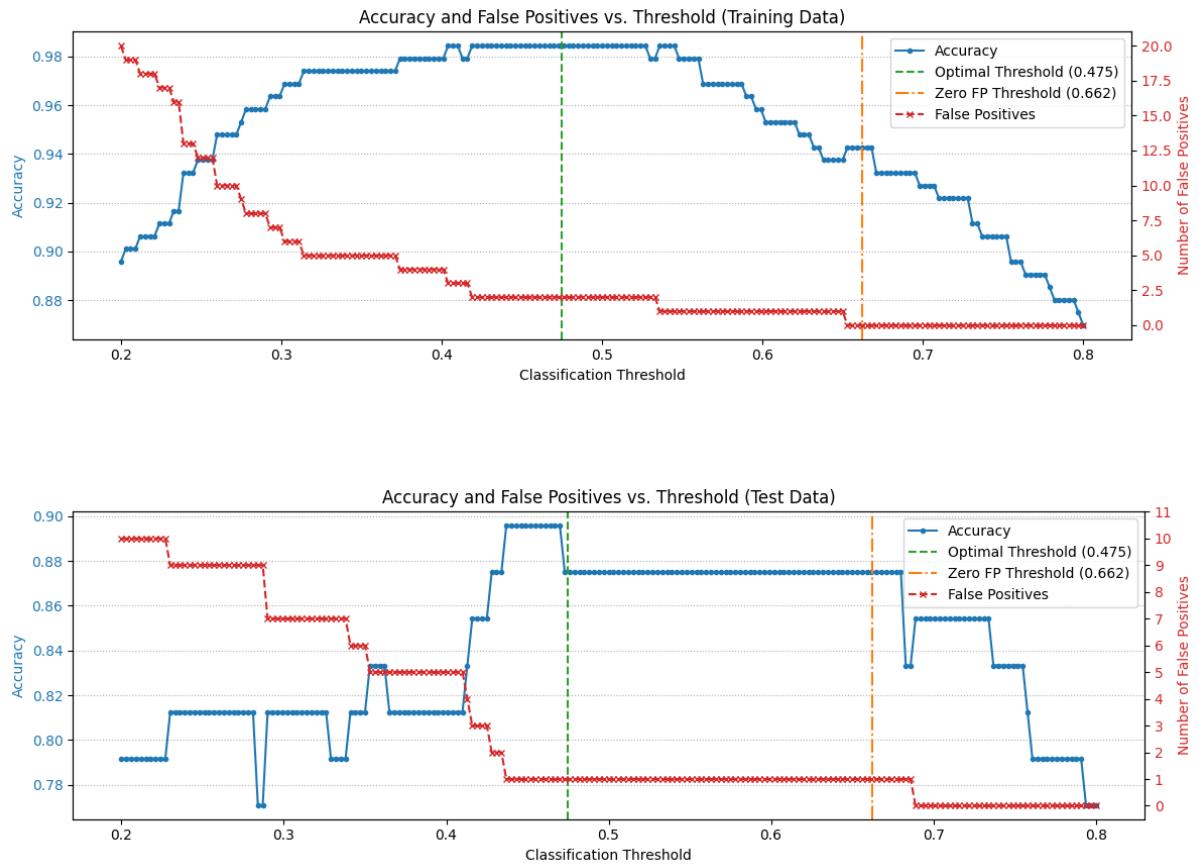


FIGURE 3.13 – Train (196 images) et Test (48 images) : Accuracy et faux positifs pour seuils  $\tau \in [0.2, 0.8]$

TABLE 3.1 – Performance

Type de Seuil	Seuil	Train (196 images)					Test (48 images)				
		Accuracy	TN	FP	FN	TP	Accuracy	TN	FP	FN	TP
Optimal	0.4750	0.9844	110	2	1	79	0.8750	26	1	5	16
Zero FP	0.6622	0.9427	112	0	11	69	0.8750	26	1	5	16

Nous arrivons à atteindre une performance acceptable  $\approx 88\%$ , indiquant que le classifieur trouve un critère acceptable pour différencier les réponses correctes des réponses incorrectes, en se servant seulement de la liste des probabilités de la réponse. On constate cependant qu'il y a un faux positif dans les données test, indiquant qu'il faut plus de données ou un seuil plus élevé pour mieux généraliser.

Voici quelques exemples sur les données de test avec l'objet, la réponse générée, la vraie classification, et les prédictions avec les deux seuils :

Image	Object	Suffix Text	GT	Pred(Opt)	Pred(0FP)	(C=1)
03.jpg	mug	coffee cup.	1	1	1	0.9853
06.jpg	sponge	piece of toast.	0	0	0	0.1383
26.jpg	controller	video game controller.	1	0	0	0.4708
36.jpg	mouse	camera.	0	0	0	0.0070

Nous pourrons ensuite tester ce classifieur en le connectant au VLM en analysant ces réponses. Avec ce classifieur, une seule réponse du VLM suffit.

## Conclusion

Le classifieur à réussi à trouver un critère intéressant mais nos données sont limitées et on ne peut pas obtenir de résultat parfait pour la classification des réponses. En particulier, les données ont été augmentés artificiellement ici et il peut donc y avoir du overfitting. Il est également recommandé de monter la valeur du seuil pour être plus sûr de ne pas avoir de faux positifs pendant l'utilisation du classifieur pour l'usage en pratique du classifieur. Cela reste une bonne heuristique pour évaluer la confiance de l'identification, ce qui est utile en robotique, surtout dans notre cas en autonomie supervisée, où l'humain ne va pas forcément corriger le robot.

Nous pouvons utiliser ce classifieur pour identifier des objets en lui donnant des images correspondant à différents points de vue, jusqu'à ce qu'il est assez 'sûr' de son identification. Un code et un exemple ont été fournis en annexe, qui simule cette interaction.

### 3.3 Embeddings

Dans cette section, nous explorons l'utilisation des embeddings comme moyen alternatif d'évaluer la qualité des réponses d'un VLM. Les embeddings sont des représentations vectorielles qui capturent le sens sémantique des mots et des phrases, et permettent de quantifier la similitude entre différents concepts.

#### 3.3.1 Principe et méthodologie

Notre approche se base sur l'hypothèse que la similitude sémantique entre la réponse d'un modèle et l'objet réel peut être mesurée via la distance entre leurs embeddings respectifs. Pour tester cette hypothèse, nous avons utilisé le modèle `nomic-embed-text` via Ollama, qui génère des vecteurs d'embedding pour des textes donnés.

Nous avons exploré deux métriques principales pour quantifier la distance entre embeddings :

- **Distance euclidienne** : mesure la distance absolue entre deux vecteurs dans l'espace vectoriel. Pour deux vecteurs  $\vec{u}$  et  $\vec{v}$  de dimension  $n$ , elle est définie par :

$$d(\vec{u}, \vec{v}) = \sqrt{\sum_{i=1}^n (u_i - v_i)^2}$$

- **Distance angulaire** (ou similarité cosinus inversée) : mesure l'angle entre deux vecteurs, ce qui est souvent plus pertinent pour comparer des vecteurs de haute dimension. Pour deux vecteurs  $\vec{u}$  et  $\vec{v}$ , elle est définie par :

$$d(\vec{u}, \vec{v}) = 1 - \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \|\vec{v}\|}$$

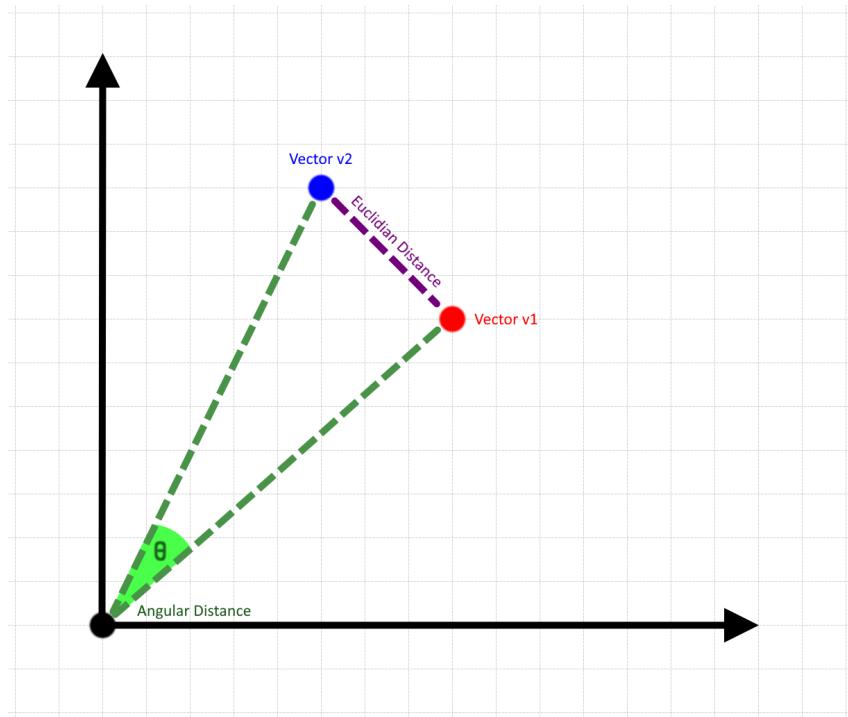


FIGURE 3.14 – Différence entre distance euclidienne et angulaire

Après suffisamment de tests, nous avons décidé d'utiliser la distance angulaire plutôt que la distance euclidienne, car elle s'est révélée un peu plus efficace pour notre cas d'utilisation. Mais pour la plupart, les deux distances donnent des résultats similaires.

Cette méthode présente plusieurs avantages par rapport à une simple correspondance de mots-clés, notamment elle permet de détecter des réponses sémantiquement correctes même lorsqu'elles utilisent des termes différents (ex : "coffee mug" vs "cup"). Elle quantifie la proximité entre concepts, ce qui permet d'évaluer le degré d'erreur d'une mauvaise classification, plutôt que de simplement la rejeter ou l'accepter.

### 3.3.2 Expérimentation et validation du concept

Pour valider l'approche, nous avons d'abord testé les embeddings sur des ensembles contrôlés de mots. Les tests ont confirmé que les embeddings capturent efficacement les relations sémantiques, mais aussi que des fois ils ne sont pas toujours parfaitement alignés avec les relations sémantiques humaines :

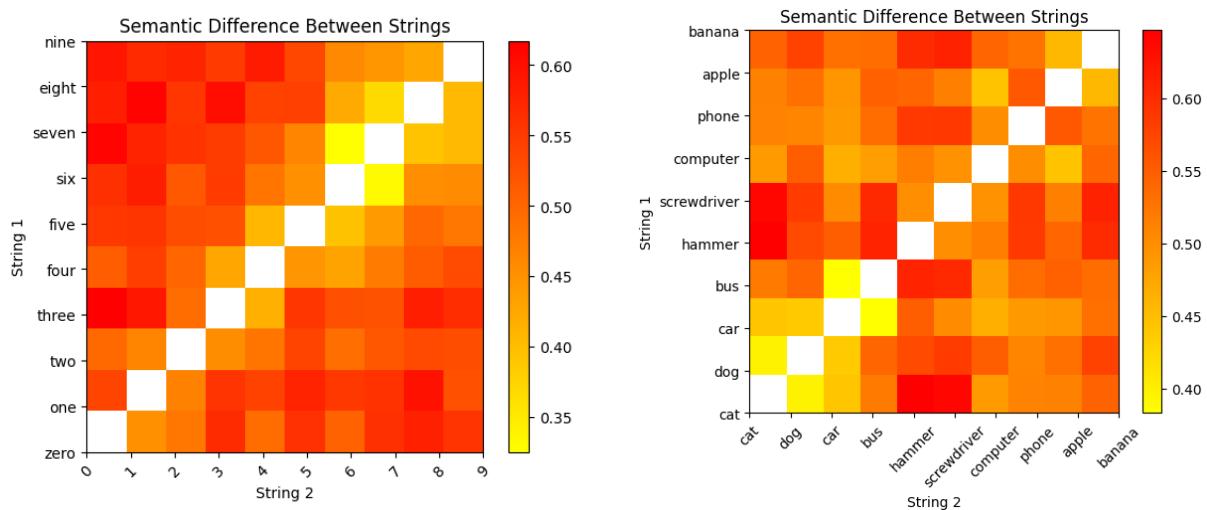


FIGURE 3.15 – Chiffre vs Mot pour le chiffre (gauche) Mots divers (droite)

Comme le montre la figure à gauche, les objets d'une même valeur sémantique (ex : "1" et "one") ont des distances très petites. Et comme le montre la figure à droite, les mots dans une même catégorie ont généralement des embeddings plus proches l'un de l'autre que des objets de catégories différentes. (ex : "cat" plus proche à "dog" que "computer")

### 3.3.3 Application à l'évaluation des réponses des VLMs

Nous avons ensuite appliqué cette approche aux réponses obtenues de différents VLMs sur notre dataset. Pour chaque image, nous avons :

1. Généré les embeddings des noms d'objets corrects
2. Généré l'embedding de la réponse fournie par le VLM
3. Calculé la distance euclidienne et angulaire entre ces embeddings, pris la distance minimale

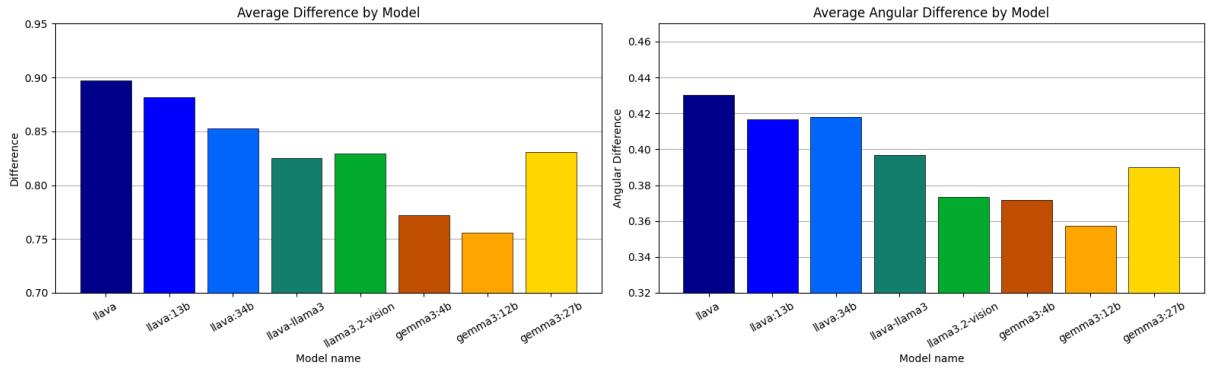


FIGURE 3.16 – La distance moyenne de chaque modèle pour le dataset d’images

On peut observer que le modèle gemma3 :12b a une distance angulaire plus faible que les autres modèles, mais ça ne signifie pas forcément qu'il est celui qui donne le plus de bonnes réponses. Pour voir cela, il faut trouver une méthode qui permet de séparer les bonnes réponses des mauvaises réponses.

### 3.3.4 Détermination du seuil optimal

Nous avons cherché à déterminer un seuil optimal permettant de distinguer les classifications correctes des incorrectes. Nous avons identifié qu'un seuil de distance angulaire de 0.3974 maximise la correspondance avec les labels manuels.

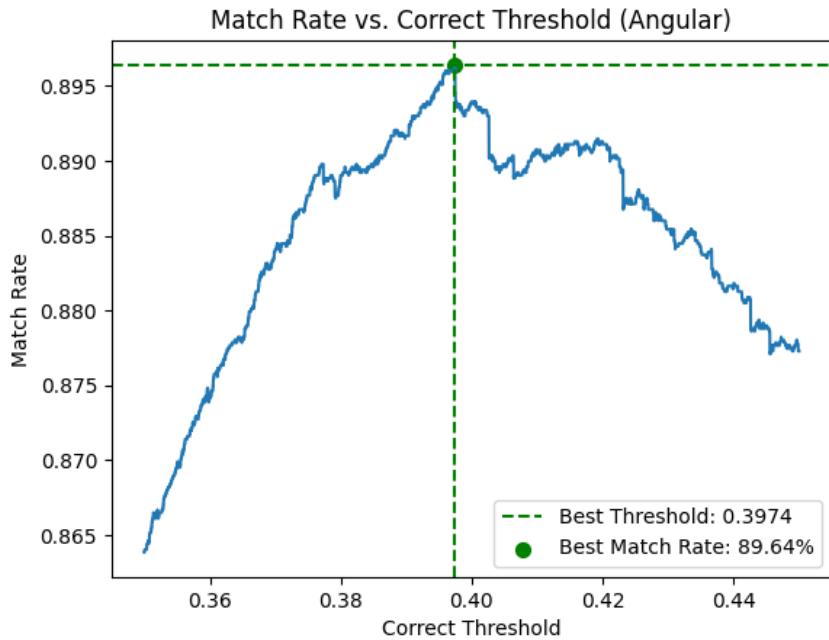


FIGURE 3.17 – Optimisation du seuil de distance angulaire

Ce calcul ne veut pas dire forcément que ce seuil est le meilleur, mais après avoir testé plusieurs seuils, nous avons trouvé que celui-ci donnait de bons résultats. En utilisant ce seuil, on peut donner un taux de réussite pour chaque modèle :

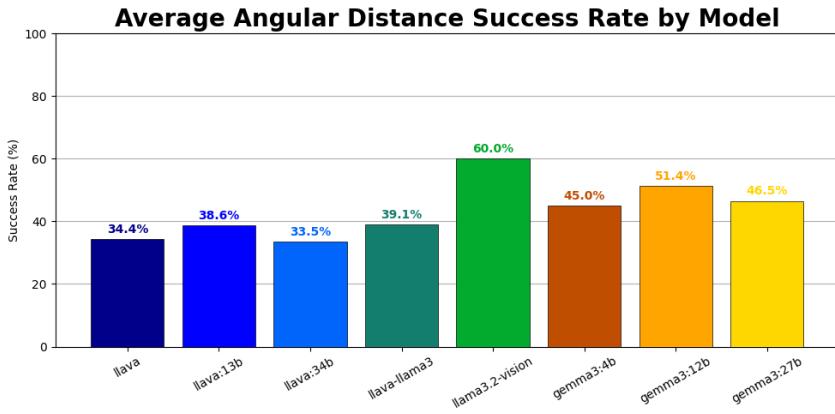


FIGURE 3.18 – Taux de réussite avec distance angulaire

On peut voir que le modèle llama3.2-vision a un taux de réussite de 60%, meilleure que gemma3:12b qui a 51%, même si sa distance angulaire moyenne était plus faible.

### 3.3.5 Analyse des cas limites

Nous avons également analysé les cas où la métrique d'embedding et la correspondance exacte de mots divergent :

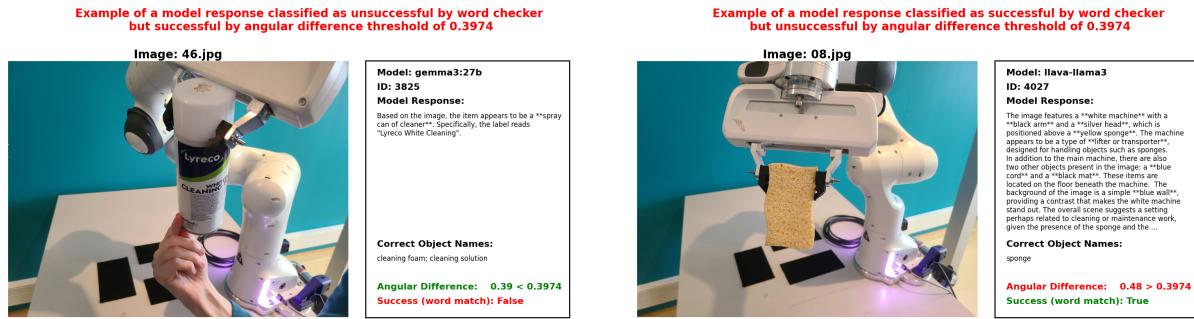


FIGURE 3.19 – Exemples de divergence

Dans l'exemple à gauche, la réponse est correcte, mais le modèle a utilisé un terme différent de celui attendu. La classification par embedding a permis de valider cette réponse malgré la divergence lexicale.

Dans l'exemple à droite, la réponse est correcte, mais le modèle a continué à générer des tokens après la réponse correcte, ce qui a conduit à une distance angulaire plus élevée, dans ce cas, la classification par correspondance de mots aurait été correcte.

Alors finalement, nous avons décidé de classifier en utilisant les deux méthodes, avec ( $\text{angular\_distance} < 0.3974$ ) OU ( $\text{word\_match}$ ) qui donne le taux de réussite finale :

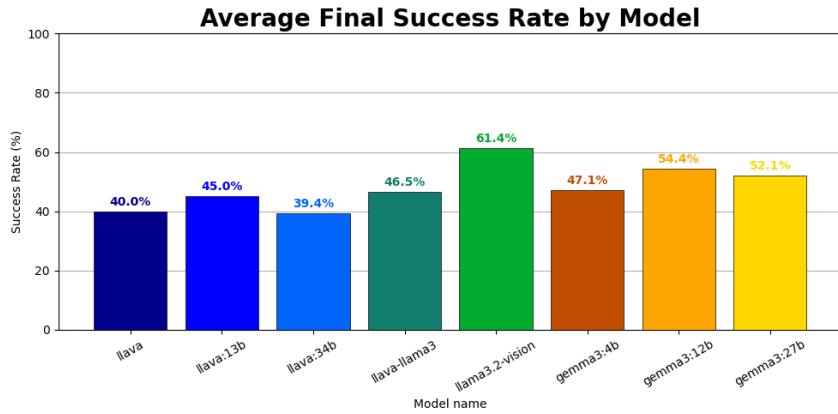


FIGURE 3.20 – Taux de réussite finale

Avec cette méthode de classification on peut aussi comparer les divers prompts

### Final Success Rate by Model and Prompt

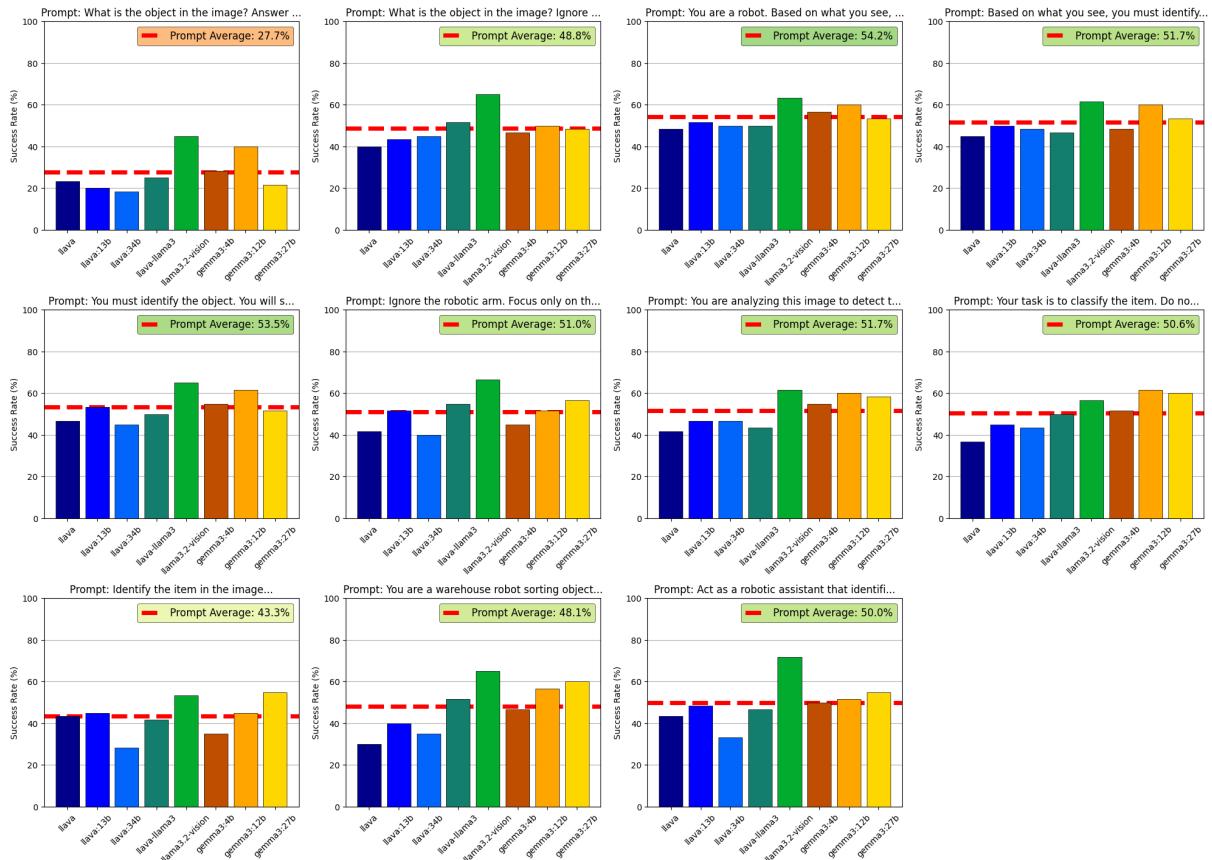


FIGURE 3.21 – Taux de réussite finale pour chaque prompt

### 3.4 Tests avec plusieurs Images

Dans cette section, nous avons exploré la possibilité d'utiliser plusieurs images pour améliorer la classification d'un objet.

On a testé sur llama3.2-vision, le meilleur modèle local qu'on a trouvé, et on a vu que des fois, c'est vraiment utile d'avoir plusieurs images à la disposition du modèle.

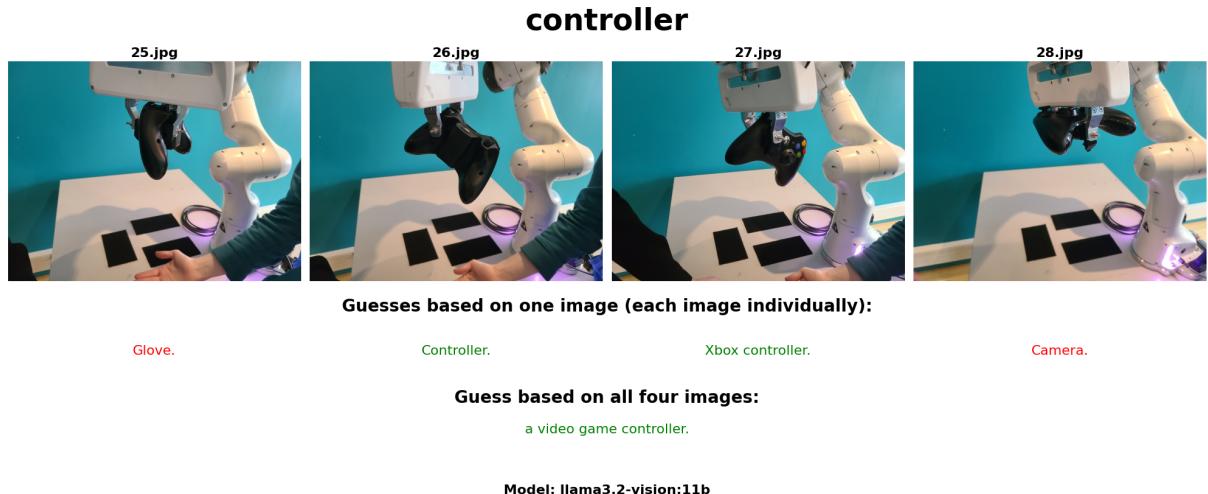


FIGURE 3.22 – Des fois oui

Mais aussi que des fois, ça ne marche pas du tout, et le modèle confond des objets qu'il arrivait à classifier correctement avec une seule image.

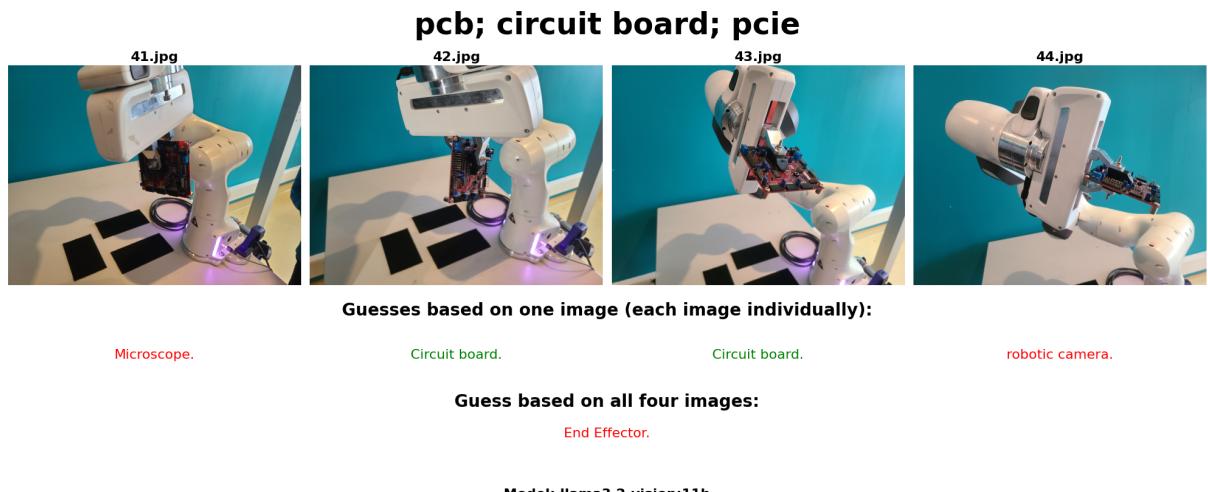


FIGURE 3.23 – Des fois non

On voit que la taux de réussite du modèle ayant vu le même objet par 4 angles n'est pas mieux en moyenne que s'il avait vu l'objet par un seul angle.



FIGURE 3.24 – Réussite ou non des réponses par rapport aux images vu par le modèle (manuellement vérifié)

### 3.5 Modèles de l'état de l'art

Après avoir mené nos expérimentations avec des modèles locaux, nous avons testé quelques-unes des images les plus problématiques (celles avec les plus faibles taux de réussite) sur des modèles de pointe tels que ChatGPT, Gemini et Claude. Ces modèles ont montré des performances nettement supérieures, parvenant à identifier correctement les objets dans la quasi-totalité des cas, y compris dans des situations où tous les modèles locaux échouaient.

Cependant, malgré leurs résultats impressionnantes, ces modèles ne sont pas adaptés à un usage embarqué. Leur taille, leur besoin de connectivité réseau, les coûts d'utilisation, les préoccupations liées à la vie privée, ainsi que leur empreinte énergétique, en font des solutions difficilement compatibles avec des systèmes robotiques autonomes ou contraints en ressources, comme ceux visés par notre étude.

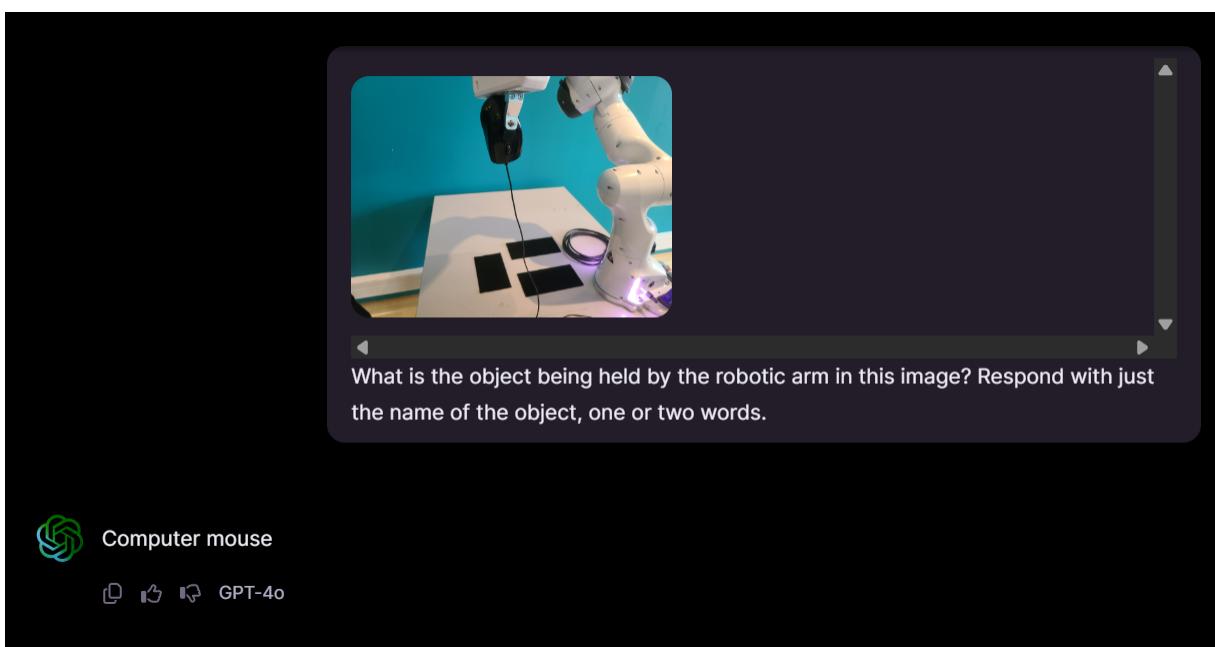


FIGURE 3.25 – ChatGPT (GPT-4o)

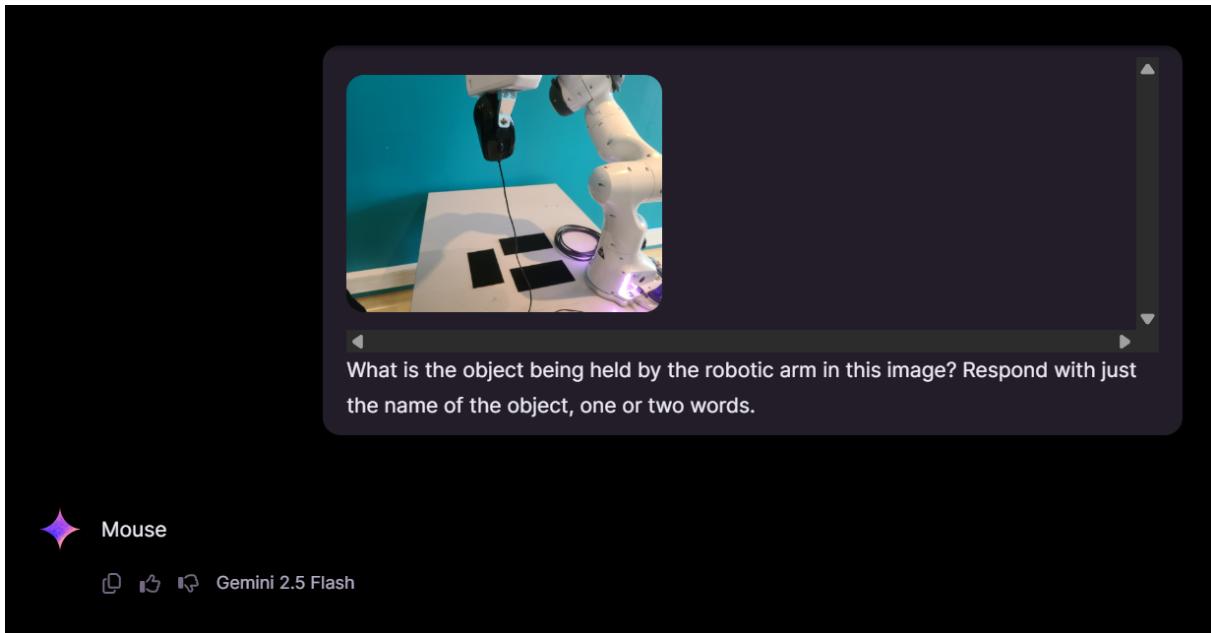


FIGURE 3.26 – Gemini (2.5 Flash)

Mais on voit aussi que même les meilleurs modèles ne sont pas parfaits, pour des images où l'objet est trop caché, ils n'arrivent pas non plus.

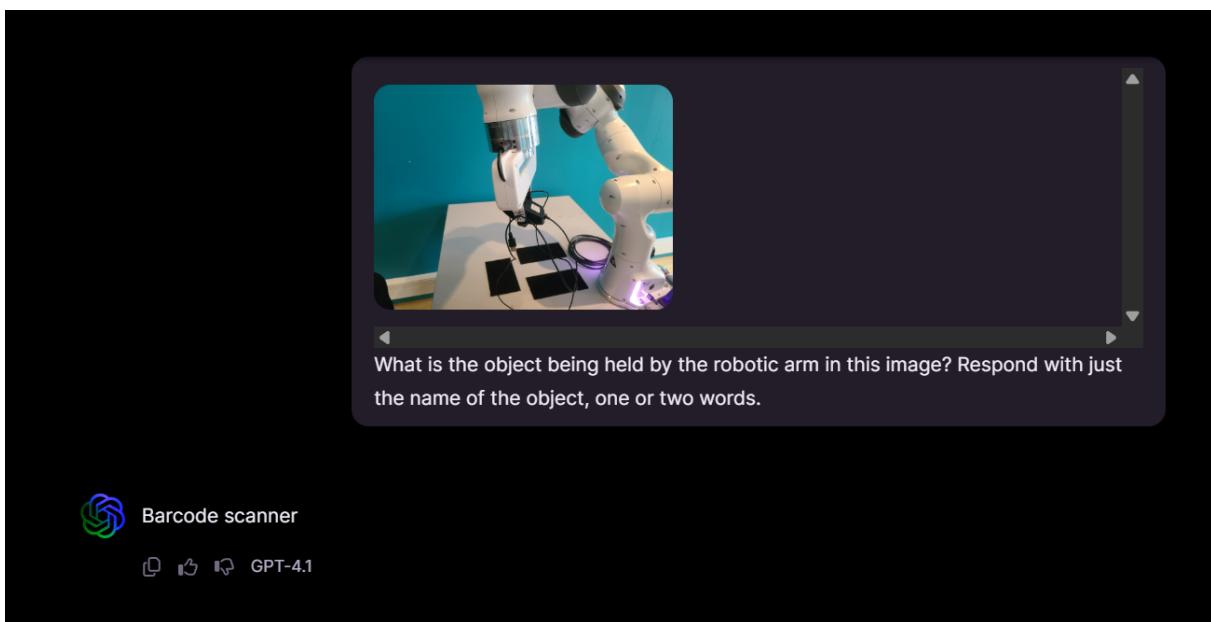


FIGURE 3.27 – ChatGPT (GPT-4.1)

# Chapitre 4

## Conclusion

### Bilan des contributions :

Dans ce projet, nous avons exploré les capacités de modèles de langage multimodaux exécutables localement pour identifier des objets à partir d'images capturées dans un contexte robotique. Nous avons d'abord étudié l'effet du prompting et des conditions visuelles sur les performances de différents modèles, avant de proposer des méthodes pour mieux estimer leur incertitude. Nous avons également introduit une approche basée sur les embeddings pour évaluer automatiquement la pertinence des réponses générées. Enfin, nous avons testé une stratégie d'agrégation multi-vues et comparé nos résultats à ceux de modèles de grande taille non embarquables.

### Forces du travail :

L'un des points forts de notre projet réside dans la répartition des tâches, chaque membre ayant exploré un aspect distinct du problème (prompting, incertitude, embeddings...). Cela nous a permis de couvrir un spectre large de questions autour de la perception robotique avec des modèles de langage visuels, tout en allant en profondeur sur chacun de ces axes. Cette diversité d'approches a contribué à une compréhension plus complète du potentiel et des limites de ces modèles dans un contexte robotique.

### Limites du travail :

Certaines limites restent toutefois présentes. Nous n'avons pas pu intégrer les différentes briques développées en un système complet de perception active, capable d'adapter dynamiquement sa stratégie en fonction de l'incertitude. Un tel système consisterait par exemple à analyser une première image, estimer la fiabilité de la prédiction, puis en cas de doute solliciter une nouvelle vue, jusqu'à atteindre un niveau de confiance suffisant ou à épuiser les observations disponibles. Les embeddings pourraient alors être utilisés pour vérifier automatiquement la cohérence des réponses avec les objets attendus. De plus, la perception dite active a été simulée à l'aide d'un jeu restreint d'images fixes prises selon différentes orientations des objets/bras robot, ce qui reste une approximation assez rigide. Enfin, la qualité des images utilisées était parfois sous-optimale : des éléments visuellement parasites (comme les 3 rectangles noirs sur la table) ont pu introduire du bruit dans les prédictions, ce qui limite la représentativité de certains résultats.

## **Retour sur les objectifs initiaux :**

L'ensemble des objectifs du cahier des charges ont été atteints : nous avons testé plusieurs modèles de vision et langage, comparé différents prompts, évalué les performances selon divers critères, et exploré des stratégies liées à la perception active via plusieurs images par objet. Nous sommes même allés plus loin que prévu sur certains aspects, comme dans l'analyse de l'incertitude ou l'utilisation d'embeddings pour l'évaluation automatique.

**Perspectives de poursuite :** Une suite naturelle à ce travail consisterait à intégrer les différentes briques développées dans un système autonome capable de fonctionner en temps réel, en enchaînant les observations en fonction de l'incertitude mesurée. Il serait également très pertinent de tester ce système sur un robot réel capable de manipuler physiquement les objets, afin d'évaluer la perception active dans un cadre dynamique et interactif plutôt que en simulant la partie active avec un set d'images prises manuellement. Pour le classifieur, il serait intéressant de récolter plus de données pour obtenir de résultats plus complets, ainsi que d'utiliser de meilleurs environnements de test comme llama.cpp (C++) pour accélérer le processus.

## **Bilan technique et retour d'expérience**

Ce projet nous a permis de mieux comprendre les enjeux concrets liés à l'utilisation de modèles de vision et langage en local, notamment dans un contexte robotique. Nous avons été confrontés à plusieurs limites techniques : les modèles testés sont relativement lourds, ce qui rend leur exécution coûteuse en ressources et parfois très lente. L'extraction d'incertitudes a également posé des difficultés : Ollama masque certaines informations internes comme les probabilités des tokens, et l'usage de llama-cpp-python s'est vu limité par des contraintes matérielles (non-fonctionnement du GPU avec llama-cpp-python). Enfin, pour les embeddings, il n'existait pas de modèle combinant vision et embedding, ce qui a empêché l'extraction directe d'un embedding d'image, mais cela n'a pas été bloquant dans notre approche, qui visait surtout à évaluer les réponses des VLMs.

Malgré ces défis, ce projet nous a permis d'explorer en profondeur un domaine technologique d'actualité, d'enrichir notre compréhension des VLMs, et de développer des compétences utiles pour la suite de notre parcours.

## **Remerciements**

Nous tenons à remercier chaleureusement nos encadrants, Emiland Garrabé et Stéphane Doncieux, pour leur disponibilité, leurs conseils pertinents tout au long du projet, et leur accompagnement régulier qui a grandement contribué à la qualité de notre travail.

# Annexe A

## Cahier des charges

# Cahier des Charges

Projet : Perception active pour la robotique basée sur des LLM

Étudiants : Thomas Marchand - Tarik Ege Eken - Victor Fleiser

Encadrants : Stéphane Doncieux - Émiland Garrabé

## 1 Introduction

Dans le cadre de ce projet, nous explorons la capacité de modèles de vision et langage à classifier des objets. L'objectif est d'évaluer plusieurs modèles et prompts, afin d'identifier les meilleures approches en termes de précision et de pertinence, puis l'exploration de méthodes pour mesurer le niveau de confiance du modèles dans son identification de l'objet, pour savoir si il faudrait que le robot déplace l'objet pour un meilleur point de vue (perception active) afin de mieux l'identifier. Ce cahier des charges définit les contraintes ainsi que la solution envisagée pour atteindre les objectifs du projet.

## 2 Objectifs

- Tester plusieurs modèles de visions pour comparer les résultats observés.
- Expérimenter avec différents prompts pour analyser leur impact sur la classification des objets.
- Expérimenter avec la perception active en fournissant des images avec d'autres angles pour essayer d'améliorer la supposition du modèle lorsqu'il n'est pas assez sûr dans son identification.
- Comparer les résultats selon plusieurs critères :
  - Exactitude de la classification.
  - Certitude de la supposition.
  - Qualité/longeur de la réponse générée.
  - Temps de réponse.
- Fournir un rapport détaillé des observations et conclusions.

## 3 Contraintes

### 3.1 Techniques

- Utilisation de modèles de vision et de langage disponibles via Ollama en Python.
- Stockage et gestion des données via GitLab et Overleaf pour les rendus.
- Quantification de l'incertitude des modèles en utilisant llama-cpp-python pour déterminer quelles images sont meilleures pour la perception active.

### 3.2 Organisationnelles

- Projet à finaliser avant le 9 mai.
- Collaboration avec les encadrants pour affiner la direction du projet ainsi que les résultats attendus.

## 4 Solution Envisagée

### 4.1 Collecte des Données

- Utilisation d'un dataset de 60 images où un bras Franka tient 12 objets différents dans différentes rotations/positions.
- Les différentes rotations/positions permettent de simuler le déplacement du bras robot pour la perception active.
- Les images peuvent également être modifiées pour tester différentes conditions afin de tester leurs impact sur les résultats.

### 4.2 Choix des Modèles

- Sélection de plusieurs modèles de vision et langage (llava, llama3.2-vision, gemma3) ainsi que plusieurs tailles des différents modèles.
- Comparaison de leurs performances sur les mêmes images et prompts.

### 4.3 Méthodologie de Test

- Application de différents prompts pour voir l'impact sur les réponses du modèle.
- Mesure des performances selon plusieurs critères (exactitude, cohérence, rapidité).

### 4.4 Analyse des Résultats

- Compilation des réponses des modèles dans des fichiers csv.
- Comparaison quantitative et qualitative des performances.
- Discussion des limites et perspectives d'amélioration.

# Annexe B

## Manuel utilisateur

Les codes créés sont séparés en 3 sections avec chacun leurs manuel d'utilisation.

### B.1 Prompting et modèles

**dossier `./prompting_and_models` sur github**

Ce dossier contient les scripts permettant de tester différentes formulations de prompts sur des modèles vision-langage, ainsi que d'évaluer leur robustesse face à des transformations d'images (flou, surexposition, etc.). Les résultats sont sauvegardés sous forme de fichiers CSV et peuvent être convertis en images pour une visualisation plus intuitive.

Librairies python requises :

- ollama : pour utiliser les modèles local installé via l'application Ollama
- pandas, matplotlib, numpy : librairies classiques
- csv, PIL : librairies pour gérer les csvs ainsi que les images créées

#### 1. partie exploration de prompts :

- main.py : Permet de tester une liste de prompts sur une liste de modèles et d'images. Les prompts sont stockés dans ressources/all\_prompts.txt, les autres entrées sont dans le fichier main.py. Les réponses des modèles ainsi que toutes les autres informations (prompt, modèle, temps d'exécution...) sont enregistrées dans output/responses.csv
- create\_CSVs\_from\_responses.py : Génère plusieurs fichiers CSV à partir de output/responses.csv :
  - output/accuracy\_matrix.csv :
    - Lignes : prompts
    - Colonnes : modèles
    - Contenu : nombre d'images correctement classifiées pour chaque couple prompt/modèle.
  - Pour chaque image : output/image\_matrices/
    - Lignes : prompts
    - Colonnes : modèles

- Contenu : True si la prédiction est correcte, False sinon.
- Pour chaque modèle : output/model\_matrices/
  - Lignes : prompts
  - Colonnes : images
  - Contenu : True si la prédiction est correcte, False sinon.
- Pour chaque prompt : output/prompt\_matrices/
  - Lignes : modèles
  - Colonnes : images
  - Contenu : True si la prédiction est correcte, False sinon.
- csv\_to\_image.py : Convertit les fichiers CSV générés en images PNG pour une visualisation plus facile.

## 2. Partie transformations d'images :

- transform\_images.py : Script permettant de générer des variantes des images originales : floutées, occlusives, surexposées, etc. Certaines transformations étudiées dans ce rapport ont été faites manuellement (rognage).
- main\_2.py : Permet de tester un prompt unique sur une liste de modèles et de dossiers d'images (contenant les différentes variantes). Les résultats sont enregistrés dans output/responses\_2.csv
- create\_CSVs\_from\_responses\_2.py : Génère plusieurs fichiers CSV à partir de output/responses\_2.csv :
  - output/accuracy\_matrix\_2.csv :
    - Lignes : transformations
    - Colonnes : modèles
    - Contenu : nombre d'images correctement classifiées par transformation/-modèle.
  - Pour chaque image : output/image\_matrices\_2/
    - Lignes : transformations
    - Colonnes : modèles
    - Contenu : True si la prédiction est correcte, False sinon.
  - Pour chaque modèle : output/model\_matrices\_2/
    - Lignes : transformations
    - Colonnes : images
    - Contenu : True si la prédiction est correcte, False sinon.
  - Pour chaque transformation : output/transformation\_matrices\_2/
    - Lignes : modèles
    - Colonnes : images
    - Contenu : True si la prédiction est correcte, False sinon.
- csv\_to\_image\_2.py : Convertit les fichiers CSV générés en images PNG pour une visualisation plus facile.

## **Important :**

Si vous souhaitez réinitialiser tous les résultats, il faut supprimer manuellement le contenu du dossier `output/`. Si vous souhaitez ajouter/modifier/supprimer des réponses enregistrées, vous pouvez modifier `output/responses.csv` (ou `output/responses_2.csv` pour les transformations) et il suffit de relancer le code `create_CSVs_from_responses.py` et `csv_to_image.py` pour recréer les tables(`create_CSVs_from_responses_2.py` et `csv_to_image_2.py` respectivement).

## **Résultats du projet :**

Nous avons laissé l'ensemble des résultats obtenus durant ce projet pour cette partie dans le dossier `output_archives/`, incluant :

- les réponses des modèles
- les fichiers CSV générés
- les visualisations associées

## **B.2 Incertitude**

### **dossier `./uncertainty` sur github**

Partie incertitude du code. Il y a de nombreux imports, notamment `llama-cpp-python`. Ils sont tous au début du code à vérifier. Les installation en plus nécessaires seront spécifiés pour chaque script.

NOTE : les modèles LLM sont chargés avec une fonction dans `load_ollama_models`. Le code de cette partie fonctionne avec `llava-v1.6`, mais le `ChatHandler` n'est pas adapté pour d'autres modèles. Il est donc conseillé de rester sur `llava 7b` ou `llava 13b`.

### **1. `create_probability_tree_multi.py`**

- Crée un arbre de probabilité pour plusieurs modèles et plusieurs images.
- Enregistre des graphes et les réponses générées et les place dans un fichier temporaire (`OUTPUT_BASE_DIR`).
- Crée un csv, des images d'arbres, un graphique montrant l'évolution du résultat, un fichier txt contenant les réponses et un tableau final (image) indiquant la probabilité d'avoir le mot correct ainsi que la masse de probabilité explorée pour chaque image et modèle.

#### **Utilisation :**

- `NUM_PATHS_TO_FIND` : nombre de chemins à générer. 10 est une bonne valeur pour tester. Il est recommandé de donner la même taille à `RANKS_K`.
- `MODEL_CONFIGS` : modèles à utiliser depuis ollama.
- Les images à utiliser doivent être mises dans `IMAGE_FOLDER = dataset_vlm_0328_chosen`.
- Les mots "corrects" doivent être indiqués dans `TARGET_WORDS_LIST`. Chaque image contiendra une liste de mots.
- Les libraries spécifiées dans le code ainsi que `graphviz` doivent être installés.
- ollama doit être installé car le code cherche directement les modèles ollama (testé sous Windows). En cas de problème, vous pouvez donner `model_path` et `mmpobj_path` manuellement.

- `load_ollama_models.py` doit être présent.

## 2. `suffix_probas.py`

- Crée un arbre de probabilité pour plusieurs modèles et plusieurs images après un préfix (caché).
- Crée un csv et un tableau final (image) indiquant des statistiques sur le suffix de la réponse donné par le VLM.

### **Utilisation :**

- NUM\_PATHS\_TO\_FIND : nombre de chemins à générer. 3 est une bonne valeur par défaut, toutes les réponses seront visibles. Il est recommandé de donner la même taille à RANKS\_K.
- TOP\_K\_FOR\_PREFIX\_SEARCH : les top k premiers tokens à regarder pour match avec notre préfixe. Si le préfixe n'est pas détecté (rare), augmenter cette valeur.
- MODEL\_CONFIGS : modèles à utiliser depuis ollama.
- Les images à utiliser doivent être mises dans IMAGE\_FOLDER = `dataset_vlm_0328_chosen`.
- Les mots "corrects" doivent être indiqués dans TARGET\_WORDS\_LIST. Chaque image contiendra une liste de mots.
- Les libraries spécifiées dans le code doivent être installés.
- ollama doit être installé car le code cherche directement les modèles ollama (testé sous Windows). En cas de problème, vous pouvez donner `model_path` et `mmpoj_path` manuellement.
- `load_ollama_models.py` doit être présent.

## 3. `probs_to_classifier.py`

- Crée et entraîne un classifieur depuis un csv pour classifier les réponses de VLM en se basant sur les probabilités de la réponse (suffixe).
- Enregistre un fichier `joblib` réutilisable.

### **Utilisation :**

- CSV\_FILE\_PATH : fichier csv à utiliser. Il est compatible avec le csv output de `2-suffix_probas.py`.
- TEST\_SET\_SIZE : proportion des réponses à utiliser pour le split test.

## 4. `identify_with_confidence`

- Obtient une réponse du VLM en identifiant un objet depuis plusieurs images, ainsi que l'incertitude donnée par le classifieur.
- Elle simule le robot qui redemande des images jusqu'à ce qu'elle soit assez sûre de son identification d'objet.
- (Attention, la classification peut-être fausse, même avec le seuil Zero False Positives.)

### **Utilisation :**

- IMAGE\_DIR : chemin du fichier contenant les images.
- IMAGE\_NAMES : noms d'images à utiliser pour l'objet à identifier (pour simuler un robot prenant plusieurs images).
- MODEL\_NAME, MODEL\_PATH, MMPROJ\_PATH = modèle LLM utilisé, spécifier manuellement des modèles .gguf si ollama non installé ou ne fonctionne pas (testé sous Windows 11).

- CONFIDENCE\_TARGET\_TYPE : type du seuil à utiliser pour définir la confiance nécessaire pour classifier comme "Sûr".
- CLASSIFIER\_MODEL\_PATH = chemin du classifieur .joblib. Il doit être entraîné sur le même type d'image, les mêmes features et le même modèle pour obtenir une performance acceptable.

## B.3 Embeddings

dossier `./embedding` sur [github](#)

Cette partie contient des scripts et notebooks pour explorer les embeddings.

**Librairies python requises :**

- `ollama` : pour accéder aux modèles locaux, notamment le modèle embedding `nomic-embed-text`
- `numpy` : pour les calculs
- `matplotlib.pyplot` : pour les visualisations
- `pandas` : pour manipuler les données tabulaires
- `PIL` : pour charger et afficher des images

**Fichiers requises :**

- `responses_with_final_success.csv` : Le fichier csv contient les vecteurs déjà calculés, pour éviter la perte du temps, faut le télécharger pour gagner accès aux données
- `images` : Le dossier `images` contient des images qui sont utilisés dans les démonstrations

**Principales fonctionnalités :**

- Extraction d'embeddings à partir de textes avec des modèles Ollama
- Calcul de similarité entre concepts (distance euclidienne et angulaire)
- Visualisation des relations sémantiques entre mots
- Analyse comparative des réponses de modèles VLM
- Évaluation de la qualité des prédictions par méthodes d'embedding

**Utilisation :**

Le code principal est fourni sous forme d'un notebook Jupyter (`embedding_test.ipynb`) qui contient tout le code (`embedding.ipynb`) la version finale qui contient tout les parties importants pour la démonstration, et qui est plus facile à lire et exécuter.

Après avoir téléchargé les fichiers et bibliothèques pertinents, il est suffisant de simplement lancer tout le fichier `embedding.ipynb`, il existe quelques lignes qui sont prêts à re-exécuter plusieurs fois pour voir des exemples sur plusieurs images ou prompts, comme les parties utilisant `compare_model_responses()` et `full_guesser()`

**Fonctions principales :**

- `get_vector(model, text)` : obtient l'embedding d'un texte depuis un modèle
- `get_difference(v1, v2)` : calcule la distance euclidienne entre deux vecteurs

- `get-angular-difference(v1, v2)` : calcule la distance angulaire (cosinus) entre deux vecteurs
- `compress_to_2d(vector)` : compresse un vecteur en 2D pour visualisation

#### Fonctions de démonstrations :

- `plot_word_differences()` : visualise les différences entre listes de mots
- `analyze_word_and_wordlist()` : analyse la proximité sémantique entre un mot et une liste
- `compare_model_responses()` : comparaison des réponses des différents modèles sur le même prompt et image
- `full_guesser()` : démonstration de l'évaluation finale du réponse d'un prompt

**Note :** Pour exécuter les analyses, vous aurez besoin d'un modèle d'embedding installé via Ollama, de préférence `nomic-embed-text`.

## Annexe C

## Exemples supplémentaires de résultats

## C.1 Prompting et Modèles

Voici quelques exemples des matrices créées et exploitées pour les observations du rendu. Ces matrices sont disponibles pour tout les modèles, toutes les images, tout les prompts, et toutes les transformations dans le dossier `./prompting_and_models/output_archives`.

Note : le texte dans les cases est trop petit pour être lu sur le rendu

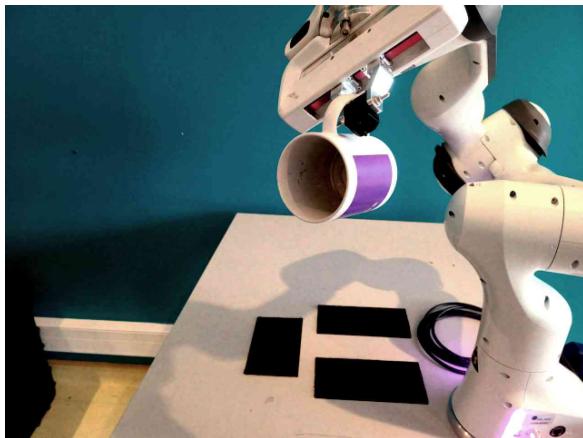


FIGURE C.1 – Ensemble des résultats pour l'image 2 (chaque ligne correspond à un prompt, chaque colonne correspond à un modèles)

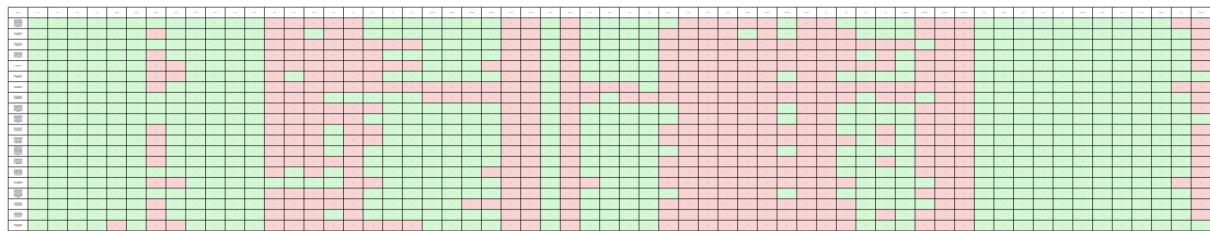


FIGURE C.2 – Ensemble des résultats pour le modèle llama3.2-vision (chaque ligne correspond à un prompt, chaque colonne correspond à une image)

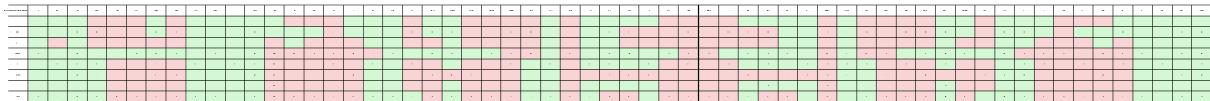


FIGURE C.3 – Ensemble des résultats pour le prompt sélectionné dans le rapport (chaque ligne correspond à un modèle, chaque colonne correspond à une image)

## C.2 Incertitude

### C.2.1 Arbre de probabilités

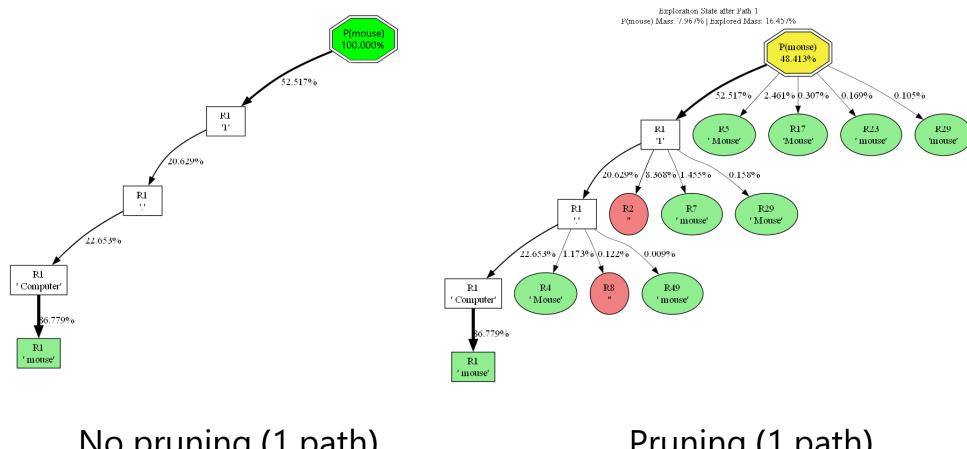


FIGURE C.4 – Exemple d'un arbre à 1 chemin, sans et avec élagage.

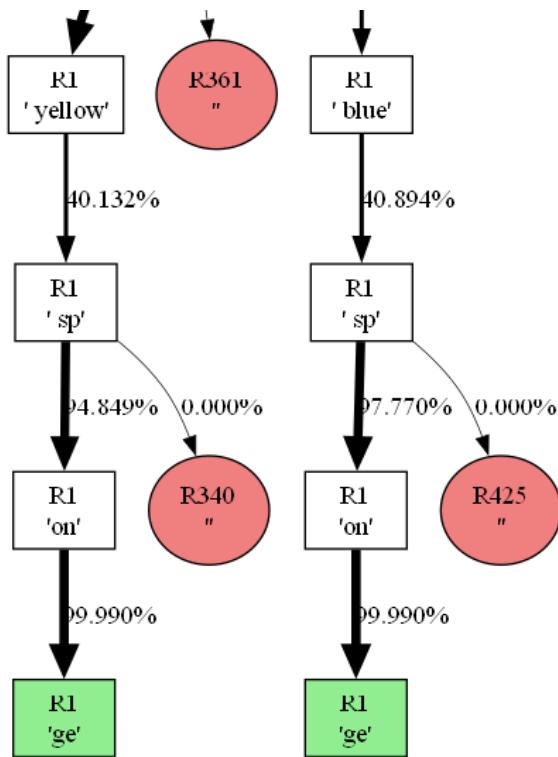


FIGURE C.5 – Arbre montrant comment le mot **sponge** est décomposé en plusieurs tokens.

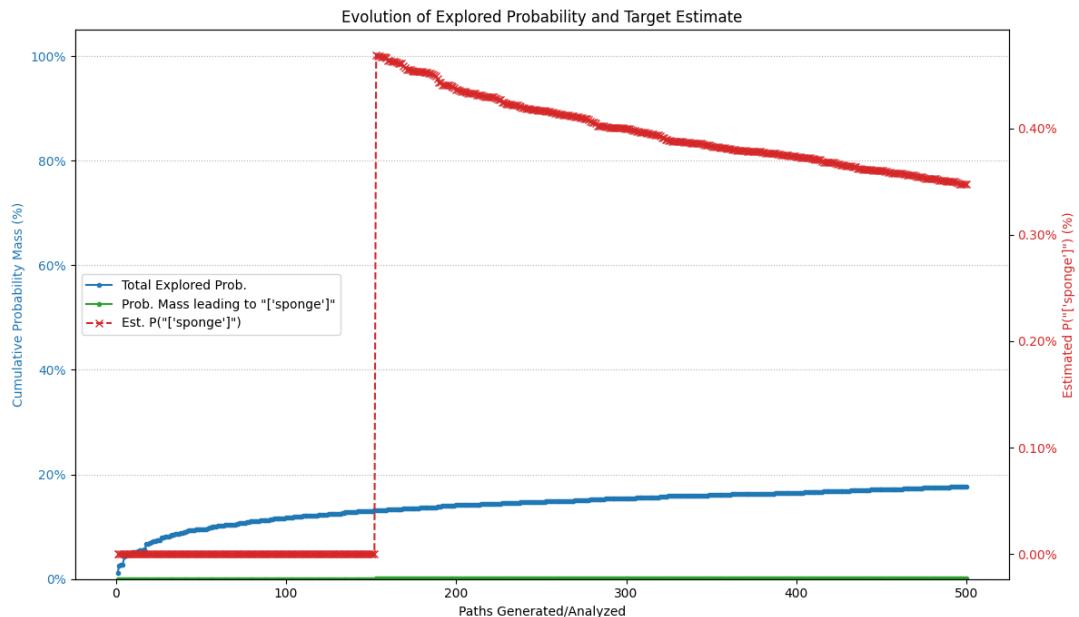


FIGURE C.6 – Évolution de la probabilité du mot et exploration montrant la limitation : Même après 500 chemins, il n'y a eu qu'une seule réponse contenant le mot **sponge**. Les mots sur plusieurs tokens ne sont pas élagués directement car le processus d'élagage ne voit que **sp**

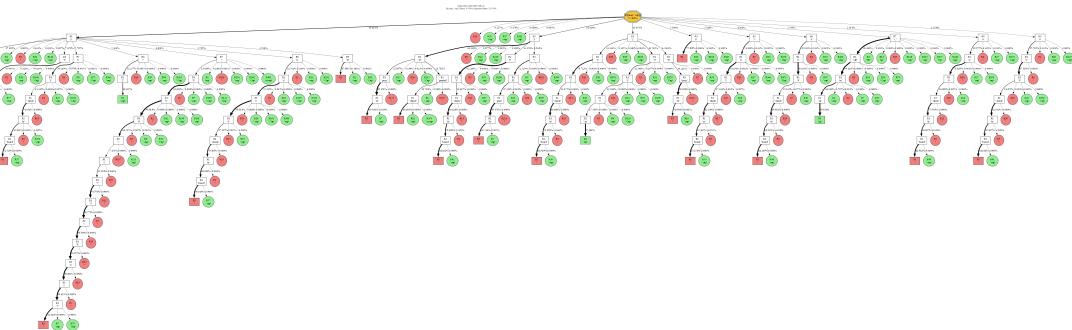


FIGURE C.7 – Exemple d'un arbre avec 20 chemins générés.

### C.2.2 Exploration de suffixe

The image is a dense grid of 64 columns by 64 rows of small square images, each representing a different object or scene from the Microsoft Common Objects in Context (COCO) dataset. The objects include a wide variety of items such as people in different poses, dogs, cats, birds, cars, trucks, bicycles, and numerous other household and outdoor items. Each image shows a specific instance of the object, often with a bounding box drawn around it to indicate its location.

FIGURE C.8 – Résultat de 3 réponses par image sur 60 images, avec des statistiques sur la probabilité de tokens.

### C.2.3 Classification selon le suffixe

```
===== Loading Classifier from: 3-classifier_llava7b.joblib =====
Will stop if confidence >= zero_fp_threshold (66.22%)

===== Processing Image 1/5: dataset_vlm_0328\01.jpg =====
    coffee mug.
Classifier Confidence (in 'Correctness'): 65.05% (< 66.22%). Requesting next image...

===== Processing Image 2/5: dataset_vlm_0328\02.jpg =====
    cup.
Classifier Confidence (in 'Correctness'): 28.86% (< 66.22%). Requesting next image...

===== Processing Image 3/5: dataset_vlm_0328\03.jpg =====
    coffee cup.
Classifier Confidence (in 'Correctness'): 98.40% (>= 66.22%). Stopping here.

=====
===== FINAL ROBOT CONCLUSION =====
Robot is SURE. Identified Object:
coffee cup.
Confidence: 98.40% (Target: 66.22%)

Suffix Token Probabilities for the confident identification:
Token: ' coffee', P: 45.287%
Token: ' cup', P: 64.439%
Token: '.', P: 92.111%
=====
```

FIGURE C.9 – Résultat d'exécution de `idenfify_with_confidence`.

Exemple d'un VLM qui classe un objet avec 5 points de vues différents. Il est sûr de son identification au bout de la 3e réponse.