



## Application Mobile d'Accessibilité Visuelle Image Captioning & Object Detection

---

*Réalisé par :*  
RAFI Mohamed Anas

*Encadré par :*  
Dr. AOURAGHE Ibtissame

# Résumé

L'essor des technologies de vision par ordinateur et de traitement du langage naturel ouvre de nouvelles perspectives pour le développement d'applications intelligentes. La génération automatique de descriptions textuelles à partir d'images, connue sous le terme *Image Captioning*, représente un défi technique majeur situé à l'intersection de ces deux domaines. Ce projet s'inscrit dans le cadre d'un stage de fin d'études visant à concevoir et implémenter un système complet de vision-langage.

La conception d'un tel système soulève plusieurs défis techniques. D'une part, l'extraction de caractéristiques visuelles pertinentes nécessite l'utilisation de réseaux de neurones convolutifs profonds. D'autre part, la génération de séquences textuelles cohérentes requiert des architectures récurrentes sophistiquées. La question centrale est la suivante : comment intégrer efficacement ces composants au sein d'une application mobile fonctionnelle, tout en maintenant des performances d'inférence acceptables pour une utilisation interactive ?

Nous proposons une architecture client-serveur privilégiant la précision via l'exploitation de modèles massifs inaccessibles aux processeurs mobiles. Le système repose sur un pipeline **Encoder-Decoder** combinant un encodeur **InceptionV3** et un décodeur **LSTM** enrichi du mécanisme d'**attention de Bahdanau**. Cette architecture est complétée par un module de détection d'objets **YOLOv12** permettant une approche de **Detection-Guided Captioning**. L'entraînement des modèles a été réalisé sur la plateforme **Kaggle** exploitant des GPU NVIDIA Tesla, avant déploiement sur un serveur d'inférence **FastAPI**. L'interface mobile est développée en **Flutter** avec persistance **Firebase Firestore**.

Les expérimentations démontrent des performances conformes aux modèles de référence de même architecture, avec des scores **BLEU-1** de l'ordre de 0.55–0.60 et des temps d'inférence d'environ **260 ms** sur accélérateur Apple Silicon. Les visualisations d'attention confirment le bon fonctionnement du mécanisme de focalisation dynamique, révélant les régions visuelles exploitées pour la génération de chaque mot.

# Abstract

The rise of computer vision and natural language processing technologies opens new perspectives for developing intelligent applications. Automatic generation of textual descriptions from images, known as *Image Captioning*, represents a major technical challenge at the intersection of these two fields. This project is part of an end-of-studies internship aimed at designing and implementing a complete vision-language system.

The design of such a system raises several technical challenges. On one hand, extracting relevant visual features requires the use of deep convolutional neural networks. On the other hand, generating coherent textual sequences requires sophisticated recurrent architectures. The central question is : how to effectively integrate these components within a functional mobile application while maintaining acceptable inference performance for interactive use ?

We propose a client-server architecture favoring precision through the exploitation of massive models inaccessible to mobile processors. The system relies on an **Encoder-Decoder** pipeline combining an **InceptionV3** encoder and an **LSTM** decoder enhanced with the **Bahdanau attention** mechanism. This architecture is complemented by a **YOLOv12** object detection module enabling a **Detection-Guided Captioning** approach. Model training was performed on the **Kaggle** platform using NVIDIA Tesla GPUs, before deployment on a **FastAPI** inference server. The mobile interface is developed in **Flutter** with **Firebase Firestore** persistence.

Experiments demonstrate performance consistent with reference models of the same architecture, achieving **BLEU-1** scores around 0.55–0.60 with inference times of approximately **260 ms** on Apple Silicon accelerator. Attention visualizations confirm the proper functioning of the dynamic focusing mechanism, revealing the visual regions exploited for generating each word.

# Table des matières

<b>Résumé</b>	<b>1</b>
<b>Abstract</b>	<b>2</b>
<b>1 Introduction Générale</b>	<b>1</b>
<b>2 Cadrage du Projet</b>	<b>2</b>
2.1 Périmètre Fonctionnel . . . . .	2
2.1.1 Fonctionnalités du Système . . . . .	2
2.1.2 Justification de l'Architecture Client-Serveur . . . . .	2
2.2 Spécifications Techniques . . . . .	3
2.2.1 Stack Technologique . . . . .	3
2.2.2 Environnement d'Entraînement . . . . .	3
2.2.3 Architecture des Modèles . . . . .	3
2.2.4 Interface de Programmation (API) . . . . .	3
2.3 Contraintes et Exigences . . . . .	4
2.3.1 Exigences de Performance . . . . .	4
2.3.2 Exigences de Qualité . . . . .	4
2.4 Critères de Validation . . . . .	4
<b>3 État de l'Art : Fondements Théoriques</b>	<b>5</b>
3.1 Réseaux de Neurones Convolutifs (CNN) . . . . .	5
3.1.1 Principe de la Convolution . . . . .	5
3.1.2 Hiérarchie des Représentations . . . . .	5
3.1.3 Architecture InceptionV3 . . . . .	5
3.1.3.1 Principe des Modules Inception . . . . .	5
3.1.3.2 Architecture Complète d'InceptionV3 . . . . .	6
3.2 Réseaux Récurrents (RNN/LSTM) . . . . .	7
3.2.1 Modélisation Séquentielle . . . . .	7
3.2.2 Architecture LSTM . . . . .	7
3.3 Mécanisme d'Attention de Bahdanau . . . . .	8
3.3.1 Motivation . . . . .	8
3.3.2 Calcul des Scores d'Attention . . . . .	8
3.3.3 Normalisation par Softmax . . . . .	8
3.3.4 Vecteur de Contexte . . . . .	9
3.4 Détection d'Objets : YOLO . . . . .	9
3.4.1 Paradigme One-Stage . . . . .	9
3.4.2 Évolution de la Famille YOLO . . . . .	9
3.5 Paradigme Encoder-Decoder . . . . .	9
3.5.1 Décodage par Beam Search . . . . .	9
<b>4 Conception et Architecture</b>	<b>11</b>
4.1 Architecture Globale du Système . . . . .	11
4.1.1 Vue d'Ensemble . . . . .	11
4.1.2 Responsabilités des Couches . . . . .	11
4.2 Pipeline de Traitement des Images . . . . .	12
4.2.1 Flux de Données . . . . .	12
4.3 Architecture Detection-Guided Captioning . . . . .	12
4.3.1 Principe de Fusion . . . . .	12
4.3.2 Mécanisme d'Intégration . . . . .	12

4.4	Architecture de l'Application Mobile . . . . .	13
4.4.1	Organisation en Couches . . . . .	13
4.4.2	Pattern Service Layer . . . . .	13
4.5	Pipeline d'Entraînement . . . . .	13
4.5.1	Environnement Kaggle . . . . .	13
4.5.2	Flux d'Entraînement . . . . .	14
4.5.3	Hyperparamètres d'Entraînement . . . . .	14
4.6	Diagramme de Séquence . . . . .	14
<b>5</b>	<b>Implémentation et Résultats</b>	<b>16</b>
5.1	Stack Technique . . . . .	16
5.1.1	Technologies Utilisées . . . . .	16
5.1.1.1	Flutter . . . . .	16
5.1.1.2	FastAPI . . . . .	16
5.1.1.3	Python . . . . .	17
5.1.1.4	PyTorch . . . . .	17
5.1.1.5	YOLO (Ultralytics) . . . . .	17
5.1.1.6	Firebase . . . . .	18
5.1.2	Dépendances Backend . . . . .	18
5.2	Interfaces Utilisateur . . . . .	18
5.2.1	Écran d'Accueil . . . . .	19
5.2.2	Écran de Détection d'Objets . . . . .	20
5.2.3	Écran de Génération de Légende . . . . .	21
5.2.4	Écran d'Analyse Combinée . . . . .	22
5.2.5	Écran Historique . . . . .	23
5.3	Résultats Expérimentaux . . . . .	24
5.3.1	Scores BLEU . . . . .	24
5.3.2	Courbes d'Apprentissage . . . . .	25
5.3.3	Analyse du Surapprentissage . . . . .	25
5.3.4	Exemples de Prédictions . . . . .	26
5.3.5	Visualisation de l'Attention . . . . .	26
5.3.6	Performances d'Inférence . . . . .	27
5.4	Comparaison avec l'État de l'Art . . . . .	28
<b>6</b>	<b>Conclusion et Perspectives</b>	<b>29</b>

# Table des figures

3.1	Structure d'un module Inception – traitement parallèle multi-échelle . . . . .	6
3.2	Architecture complète d'InceptionV3 . . . . .	6
4.1	Architecture trois couches du système AI Vision . . . . .	11
4.2	Pipeline de traitement d'une requête d'analyse . . . . .	12
4.3	Pipeline Detection-Guided Captioning . . . . .	12
4.4	Architecture en couches de l'application Flutter . . . . .	13
4.5	Pipeline d'entraînement sur Kaggle . . . . .	14
4.6	Diagramme de séquence pour une analyse combinée . . . . .	14
5.1	Logo Flutter . . . . .	16
5.2	Logo FastAPI . . . . .	16
5.3	Logo Python . . . . .	17
5.4	Logo PyTorch . . . . .	17
5.5	Logo YOLO . . . . .	17
5.6	Logo Firebase . . . . .	18
5.7	Interface d'accueil de l'application AI Vision . . . . .	19
5.8	Interface de détection d'objets en temps réel . . . . .	20
5.9	Interface de génération de légende . . . . .	21
5.10	Interface d'analyse combinée (Detection-Guided) . . . . .	22
5.11	Interface de l'historique Firebase . . . . .	23
5.12	Évolution des scores BLEU au cours de l'entraînement . . . . .	24
5.13	Courbes de perte (loss) d'entraînement et de validation . . . . .	25
5.14	Analyse comparative train vs validation . . . . .	25
5.15	Exemples de légendes générées vs références . . . . .	26
5.16	Poids d'attention pour le premier exemple . . . . .	26
5.17	Poids d'attention pour le deuxième exemple . . . . .	27
5.18	Poids d'attention pour le troisième exemple . . . . .	27

# Liste des tableaux

2.1	Fonctionnalités du système proposé . . . . .	2
2.2	Stack technique du système . . . . .	3
2.3	Spécifications des modèles de deep learning . . . . .	3
2.4	Spécification des endpoints REST . . . . .	3
2.5	Objectifs de performance . . . . .	4
3.1	Caractéristiques techniques d'InceptionV3 . . . . .	7
3.2	Portes du LSTM et leurs rôles . . . . .	7
3.3	Évolution de l'architecture YOLO (versions récentes, modèles X) . . . . .	9
3.4	Comparaison des stratégies de décodage . . . . .	9
4.1	Responsabilités des couches architecturales . . . . .	11
4.2	Services de l'application . . . . .	13
4.3	Hyperparamètres du modèle de captioning . . . . .	14
5.1	Principales dépendances Python . . . . .	18
5.2	Benchmark d'inférence (Apple M3 Pro, MPS) . . . . .	27
5.3	Positionnement par rapport aux modèles de référence . . . . .	28

# Chapitre 1

## Introduction Générale

Le domaine de l'intelligence artificielle connaît une croissance exponentielle, portée par les avancées significatives des architectures de réseaux de neurones profonds. Parmi les applications émergentes, la génération automatique de descriptions textuelles à partir d'images — désignée par le terme anglais *Image Captioning* — constitue un défi technique majeur situé à l'intersection de la vision par ordinateur et du traitement du langage naturel.

Ce projet s'inscrit dans le cadre d'un stage de fin d'études dont l'objectif est la conception et l'implémentation d'un système complet de vision-langage. Le défi consiste à maîtriser l'ensemble du cycle de vie d'une application de Deep Learning, depuis l'entraînement des modèles jusqu'au déploiement sur une plateforme mobile fonctionnelle.

L'intérêt académique de ce travail réside dans l'intégration cohérente de plusieurs composants d'architectures avancées : un **encodeur visuel** (réseau convolutif) pour l'extraction de caractéristiques d'images, un **décodeur séquentiel** (réseau récurrent) pour la génération de texte, un **mécanisme d'attention** pour la focalisation dynamique sur les régions pertinentes, un **détecteur d'objets** pour l'enrichissement sémantique, et une **architecture distribuée** pour le déploiement mobile.

La conception d'un système d'Image Captioning soulève plusieurs défis techniques interconnectés. Comment extraire des caractéristiques visuelles suffisamment riches pour capturer la sémantique d'une scène ? Comment produire des descriptions textuelles grammaticalement correctes et contextuellement pertinentes ? Comment permettre au modèle de focaliser son attention sur les régions appropriées de l'image lors de la génération de chaque mot ? Comment combiner efficacement les informations de détection d'objets avec le pipeline de génération de légendes ? Et enfin, comment rendre ce système accessible via une application mobile performante ?

Nous nous proposons d'implémenter un modèle **Encoder-Decoder** complet pour l'Image Captioning, intégrant le mécanisme d'**attention de Bahdanau**. Ce modèle est capable d'encoder une image en représentation vectorielle via un réseau convolutif pré-entraîné, de décoder cette représentation en une séquence de mots via un réseau récurrent, et d'exploiter l'attention pour pondérer dynamiquement les caractéristiques visuelles. Nous combinons également ce module avec un détecteur d'objets **YOLOv12** afin de créer une approche de **Detection-Guided Captioning**, enrichissant le contexte sémantique disponible pour le générateur de légendes.

Le présent rapport est structuré en six chapitres. Après cette introduction générale, le deuxième chapitre définit le périmètre technique et les spécifications du projet. Le troisième chapitre expose les fondements théoriques des architectures de deep learning utilisées. Le quatrième chapitre détaille la conception et l'architecture technique du système. Le cinquième chapitre présente l'implémentation et analyse les résultats expérimentaux. Enfin, le sixième chapitre conclut le rapport et ouvre sur les perspectives d'évolution.

# Chapitre 2

# Cadrage du Projet

## Introduction

Ce chapitre a pour objectif de définir le périmètre technique du projet AI Vision, de justifier les choix architecturaux retenus, et de présenter les spécifications fonctionnelles du système proposé.

### 2.1 Périmètre Fonctionnel

#### 2.1.1 Fonctionnalités du Système

Le système AI Vision offre les fonctionnalités suivantes :

TABLE 2.1 – Fonctionnalités du système proposé

Fonctionnalité	Description
Image Captioning	Génération automatique de légendes textuelles décrivant le contenu d'une image
Object Detection	Détection et localisation d'objets parmi 80 classes du dataset COCO
Analyse Combinée	Fusion des résultats de détection et de captioning pour une description enrichie
Historique	Persistance des analyses antérieures avec stockage cloud

#### 2.1.2 Justification de l'Architecture Client-Serveur

L'architecture retenue privilégie une approche **client-serveur** pour maximiser la précision des résultats. Cette décision repose sur plusieurs considérations techniques :

1. **Modèles massifs** : Les architectures de deep learning exploitées (InceptionV3, YOLOv12x) atteignent des tailles supérieures à 1 Go, inaccessibles aux contraintes mémoire des processeurs mobiles.
2. **Précision optimale** : L'exécution sur serveur permet d'exploiter la pleine précision des modèles sans quantification ni compression, garantissant des résultats de qualité supérieure.
3. **Accélération matérielle** : Le serveur bénéficie d'accélérateurs dédiés (GPU NVIDIA, Apple Silicon MPS) offrant des performances d'inférence significativement supérieures.
4. **Évolutivité** : Cette architecture permet de mettre à jour les modèles côté serveur sans nécessiter de redéploiement de l'application mobile.

## 2.2 Spécifications Techniques

### 2.2.1 Stack Technologique

TABLE 2.2 – Stack technique du système

Composant	Technologie	Rôle
Frontend Mobile	Flutter / Dart	Interface utilisateur cross-platform
Backend API	FastAPI / Python	Serveur REST pour l'inférence
Framework DL	PyTorch	Exécution des modèles de deep learning
Détection	Ultralytics YOLO	Module de détection d'objets
Persistance	Firebase Firestore	Base de données cloud NoSQL

### 2.2.2 Environnement d'Entraînement

L'entraînement des modèles de deep learning a été réalisé sur la plateforme **Kaggle**, exploitant des GPU **NVIDIA Tesla P100/T4**. Cette infrastructure cloud offre :

- Accès gratuit à des ressources GPU performantes
  - Environnement Python pré-configuré avec les bibliothèques de deep learning
  - Stockage persistant pour les datasets et les poids des modèles
- Les modèles entraînés sont ensuite exportés et déployés sur le serveur d'inférence FastAPI.

### 2.2.3 Architecture des Modèles

TABLE 2.3 – Spécifications des modèles de deep learning

Modèle	Architecture	Pré-entraînement
Encodeur	InceptionV3	ImageNet (1.2M images)
Décodeur	LSTM + Attention Bahdanau	Flickr8k (8K images)
Détecteur	YOLOv12x	COCO (330K images)

### 2.2.4 Interface de Programmation (API)

Le serveur expose une API REST documentée permettant l'accès aux fonctionnalités d'inférence :

TABLE 2.4 – Spécification des endpoints REST

Méthode	Endpoint	Description
POST	/caption	Génère une légende textuelle pour l'image fournie
POST	/detect	Déetecte les objets présents dans l'image
POST	/analyze	Analyse combinée (détection + captioning guidé)
GET	/health	Vérifie l'état du serveur et des modèles chargés

## 2.3 Contraintes et Exigences

### 2.3.1 Exigences de Performance

TABLE 2.5 – Objectifs de performance

Opération	Objectif	Mesuré
Génération de légende	< 500 ms	~200 ms
Détection d'objets	< 100 ms	~50 ms
Analyse combinée	< 500 ms	~260 ms

### 2.3.2 Exigences de Qualité

- **Pertinence** : Les légendes générées doivent refléter fidèlement le contenu visuel de l'image
- **Cohérence grammaticale** : Les descriptions doivent être syntaxiquement correctes
- **Fiabilité** : Le système doit maintenir une disponibilité élevée

## 2.4 Critères de Validation

La validation du système repose sur les critères suivants :

1. **Fonctionnel** : L'ensemble des fonctionnalités (captioning, détection, analyse, historique) est opérationnel
2. **Performance** : Les temps de réponse respectent les objectifs définis
3. **Qualité** : Les métriques BLEU atteignent des scores comparables aux modèles de référence
4. **Déploiement** : L'application est installable sur les plateformes iOS et Android

## Conclusion

En guise de conclusion, ce chapitre a permis de définir le périmètre technique du projet AI Vision. Nous avons justifié le choix d'une architecture client-serveur privilégiant la précision, présenté le stack technologique retenu, et établi les critères de validation. Le chapitre suivant expose les fondements théoriques des architectures de deep learning utilisées.

# Chapitre 3

# État de l'Art : Fondements Théoriques

## Introduction

Ce chapitre a pour objectif de présenter les fondements théoriques des architectures de deep learning utilisées dans le projet. Nous abordons successivement les réseaux convolutifs, les modèles récurrents, le mécanisme d'attention, et les détecteurs d'objets.

### 3.1 Réseaux de Neurones Convolutifs (CNN)

#### 3.1.1 Principe de la Convolution

Les réseaux de neurones convolutifs (*Convolutional Neural Networks*, CNN) constituent l'architecture de référence pour le traitement d'images. Leur efficacité repose sur l'opération de convolution, qui permet l'extraction automatique de caractéristiques.

L'opération de convolution bidimensionnelle s'exprime par :

$$\text{Sortie}(i, j) = \sum_m \sum_n \text{Image}(i + m, j + n) \times \text{Filtre}(m, n) \quad (3.1)$$

**Explication des symboles :**

- $\text{Image}(i + m, j + n)$  : valeur du pixel à la position  $(i + m, j + n)$  dans l'image
- $\text{Filtre}(m, n)$  : poids du filtre (noyau de convolution) à la position  $(m, n)$
- $(i, j)$  : coordonnées de sortie
- $\sum$  : somme de tous les produits pixel  $\times$  poids

#### 3.1.2 Hiérarchie des Représentations

Les CNN apprennent des représentations de complexité croissante :

- **Couches superficielles** : Contours, textures
- **Couches intermédiaires** : Formes, motifs
- **Couches profondes** : Objets, scènes

#### 3.1.3 Architecture InceptionV3

Le réseau **InceptionV3** (Szegedy et al., 2016) est utilisé comme encodeur visuel dans notre système. Son innovation principale réside dans les *modules Inception* qui combinent des convolutions de différentes tailles en parallèle.

##### 3.1.3.1 Principe des Modules Inception

L'idée clé est d'appliquer simultanément plusieurs filtres de différentes tailles sur la même entrée, puis de concaténer les résultats. Cela permet de capturer des motifs à différentes échelles :

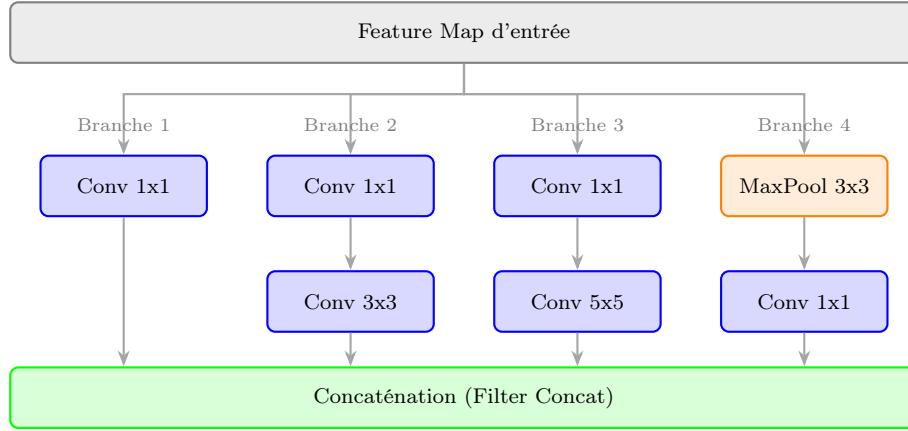


FIGURE 3.1 – Structure d'un module Inception – traitement parallèle multi-échelle

#### Rôle de chaque branche :

- **Branche 1 ( $1 \times 1$ )** : Capture les informations ponctuelles, réduit les dimensions
- **Branche 2 ( $3 \times 3$ )** : Détecte les motifs de taille moyenne
- **Branche 3 ( $5 \times 5$ )** : Capture les patterns plus larges
- **Branche 4 (MaxPool)** : Préserve les caractéristiques les plus saillantes

#### 3.1.3.2 Architecture Complète d'InceptionV3

Le réseau complet empile plusieurs modules Inception organisés en trois groupes (A, B, C), séparés par des couches de réduction qui diminuent la résolution spatiale :

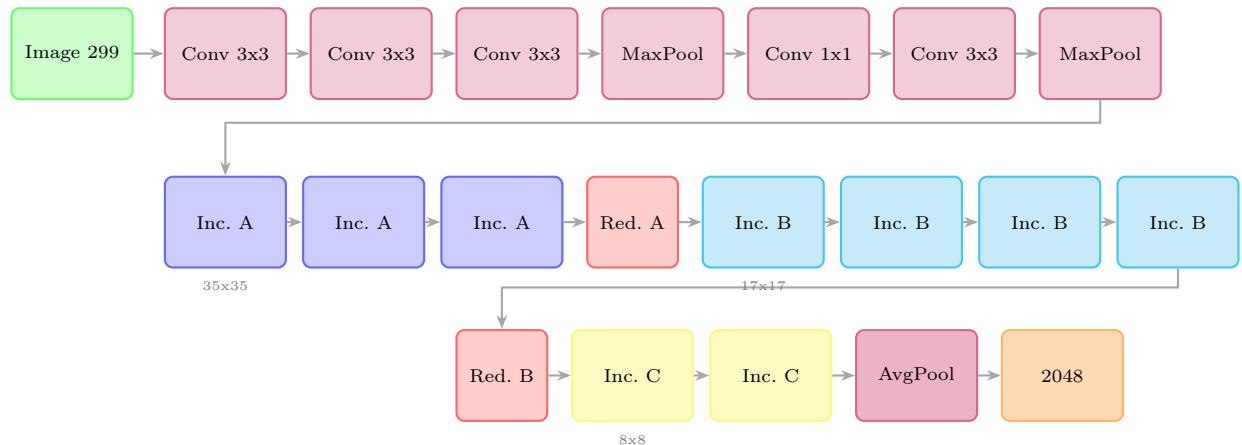


FIGURE 3.2 – Architecture complète d'InceptionV3

#### Description des composants :

- **Stem (violet)** : 7 couches convolutives initiales réduisant  $299 \times 299 \rightarrow 35 \times 35$
- **Inception A (bleu)** : 3 modules, résolution  $35 \times 35$ , 288 filtres
- **Reduction A (rouge)** : Réduit à  $17 \times 17$  avec stride 2
- **Inception B (cyan)** : 4 modules, résolution  $17 \times 17$ , 768 filtres
- **Reduction B (rouge)** : Réduit à  $8 \times 8$  avec stride 2
- **Inception C (jaune)** : 2 modules, résolution  $8 \times 8$ , 2048 filtres
- **Global AvgPool** : Moyenne spatiale  $\rightarrow$  vecteur  $1 \times 2048$

TABLE 3.1 – Caractéristiques techniques d’InceptionV3

Propriété	Valeur
Entrée	Image $299 \times 299$ pixels (RGB)
Sortie	Vecteur de 2048 dimensions
Profondeur	48 couches (sans compter pooling)
Paramètres	23.8 millions
Opérations	5.7 milliards de multiplications-additions
Pré-entraînement	ImageNet (1.2 million d’images, 1000 classes)
Top-1 Accuracy	78.8% sur ImageNet

## 3.2 Réseaux Récursifs (RNN/LSTM)

### 3.2.1 Modélisation Séquentielle

Les réseaux récurrents traitent des séquences (comme des phrases). À chaque étape  $t$ , le réseau calcule un état caché :

$$h_t = \tanh(W \times h_{t-1} + U \times x_t + b) \quad (3.2)$$

**Explication des symboles :**

- $h_t$  : état caché au temps  $t$  (mémoire du réseau)
- $h_{t-1}$  : état caché précédent (au temps  $t - 1$ )
- $x_t$  : entrée actuelle (mot actuel encodé)
- $W$  : matrice de poids pour l’état précédent
- $U$  : matrice de poids pour l’entrée
- $b$  : biais (terme constant)
- $\tanh$  : fonction d’activation (compresse les valeurs entre -1 et 1)

### 3.2.2 Architecture LSTM

Les réseaux **LSTM** (*Long Short-Term Memory*) utilisent des “portes” pour contrôler le flux d’information :

TABLE 3.2 – Portes du LSTM et leurs rôles

Porte	Rôle
Porte d’oubli $f$	Décide quoi oublier de la mémoire précédente
Porte d’entrée $i$	Décide quoi ajouter à la mémoire
Porte de sortie $o$	Décide quoi exposer comme sortie

Les équations du LSTM sont :

**1. Porte d’oubli** (quoi oublier ?) :

$$f_t = \sigma(W_f \times [h_{t-1}, x_t] + b_f) \quad (3.3)$$

**2. Porte d’entrée** (quoi ajouter ?) :

$$i_t = \sigma(W_i \times [h_{t-1}, x_t] + b_i) \quad (3.4)$$

**3. Nouvelle information candidate** :

$$\tilde{C}_t = \tanh(W_C \times [h_{t-1}, x_t] + b_C) \quad (3.5)$$

4. Mise à jour de la mémoire :

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \quad (3.6)$$

5. Porte de sortie (quoi exposer?) :

$$o_t = \sigma(W_o \times [h_{t-1}, x_t] + b_o) \quad (3.7)$$

6. État caché final :

$$h_t = o_t \odot \tanh(C_t) \quad (3.8)$$

**Explication des symboles communs :**

- $\sigma$  : fonction sigmoïde (compresse entre 0 et 1, sert de “porte”)
- $\tanh$  : fonction tangente hyperbolique (compresse entre -1 et 1)
- $\odot$  : multiplication élément par élément
- $[h_{t-1}, x_t]$  : concaténation de l'état précédent et de l'entrée
- $C_t$  : état cellulaire (mémoire à long terme)
- $W_f, W_i, W_C, W_o$  : matrices de poids (paramètres appris)
- $b_f, b_i, b_C, b_o$  : biais (termes constants)

### 3.3 Mécanisme d'Attention de Bahdanau

#### 3.3.1 Motivation

Sans attention, le modèle compresse toute l'image en un seul vecteur, perdant de l'information. L'attention permet de “regarder” différentes régions de l'image selon le mot à générer.

#### 3.3.2 Calcul des Scores d'Attention

Pour chaque région  $j$  de l'image, on calcule un score d'attention :

$$\text{score}_j = V^\top \times \tanh(W_1 \times \text{features}_j + W_2 \times \text{état\_décodeur}) \quad (3.9)$$

**Explication des symboles :**

- $\text{score}_j$  : importance de la région  $j$  pour le mot actuel
- $\text{features}_j$  : caractéristiques visuelles de la région  $j$  (vecteur de 2048 nombres)
- $\text{état\_décodeur}$  : état caché actuel du LSTM (vecteur de 512 nombres)
- $W_1, W_2$  : matrices de poids appris
- $V$  : vecteur de poids appris
- $\tanh$  : fonction d'activation
- $^\top$  : transposée (inverse lignes/colonnes)

#### 3.3.3 Normalisation par Softmax

Les scores sont convertis en poids entre 0 et 1 (sommant à 1) :

$$\alpha_j = \frac{e^{\text{score}_j}}{\sum_{k=1}^{64} e^{\text{score}_k}} \quad (3.10)$$

**Explication :**

- $\alpha_j$  : poids d'attention pour la région  $j$  (entre 0 et 1)
- $e^x$  : exponentielle de  $x$
- La somme des 64 poids  $\alpha_j$  vaut 1 (100%)
- Plus  $\alpha_j$  est grand, plus le modèle “regarde” la région  $j$

### 3.3.4 Vecteur de Contexte

Le contexte est la moyenne pondérée des caractéristiques visuelles :

$$\text{contexte} = \sum_{j=1}^{64} \alpha_j \times \text{features}_j \quad (3.11)$$

**Explication :**

- contexte : vecteur résumant “où le modèle regarde” (2048 nombres)
- On multiplie chaque région par son poids d’attention, puis on additionne
- 64 : nombre de régions spatiales (grille  $8 \times 8$ )

## 3.4 Détection d’Objets : YOLO

### 3.4.1 Paradigme One-Stage

L’architecture **YOLO** (*You Only Look Once*) détecte tous les objets en une seule passe :

1. L’image est divisée en une grille (par exemple  $13 \times 13$ )
2. Chaque cellule prédit les objets qu’elle contient
3. Une seule inférence produit toutes les détections

### 3.4.2 Évolution de la Famille YOLO

TABLE 3.3 – Évolution de l’architecture YOLO (versions récentes, modèles X)

Version	Année	mAP (COCO)	Innovation
YOLOv9x	2024	55.6%	PGI, GELAN architecture
YOLOv10x	2024	54.4%	NMS-free, dual assignments
YOLOv11x	2024	54.7%	C3k2, C2PSA attention
<b>YOLOv12x</b>	<b>2025</b>	<b>55.2%</b>	<b>Area Attention</b>

**Note :** mAP = Mean Average Precision sur le dataset COCO val2017. Les valeurs correspondent aux modèles de taille “X” (extra-large).

## 3.5 Paradigme Encoder-Decoder

L’Image Captioning combine les composants ainsi :

1. **Encodeur CNN** : Image  $\rightarrow$  64 vecteurs de 2048 nombres
2. **Attention** : Pondère les 64 régions selon le contexte
3. **Décodeur LSTM** : Génère les mots un par un

### 3.5.1 Décodage par Beam Search

Au lieu de choisir le mot le plus probable à chaque étape (glouton), le **Beam Search** garde les  $k$  meilleures phrases candidates :

TABLE 3.4 – Comparaison des stratégies de décodage

Stratégie	Complexité	Qualité
Glouton	Rapide	Moyenne
Beam Search ( $k=3$ )	$3 \times$ plus lent	Meilleure

## Conclusion

En guise de conclusion, ce chapitre a présenté les fondements théoriques des architectures exploitées : CNN pour l'extraction de caractéristiques visuelles, LSTM pour la génération séquentielle, attention de Bahdanau pour la focalisation dynamique, et YOLO pour la détection d'objets. Le chapitre suivant détaille la conception et l'architecture du système.

# Chapitre 4

# Conception et Architecture

## Introduction

Ce chapitre a pour objectif de présenter la conception détaillée du système AI Vision. Nous exposons l'architecture globale, les pipelines de traitement, et les flux de données entre les différents composants.

### 4.1 Architecture Globale du Système

#### 4.1.1 Vue d'Ensemble

Le système adopte une architecture **client-serveur** distribuée, organisée en trois couches distinctes :

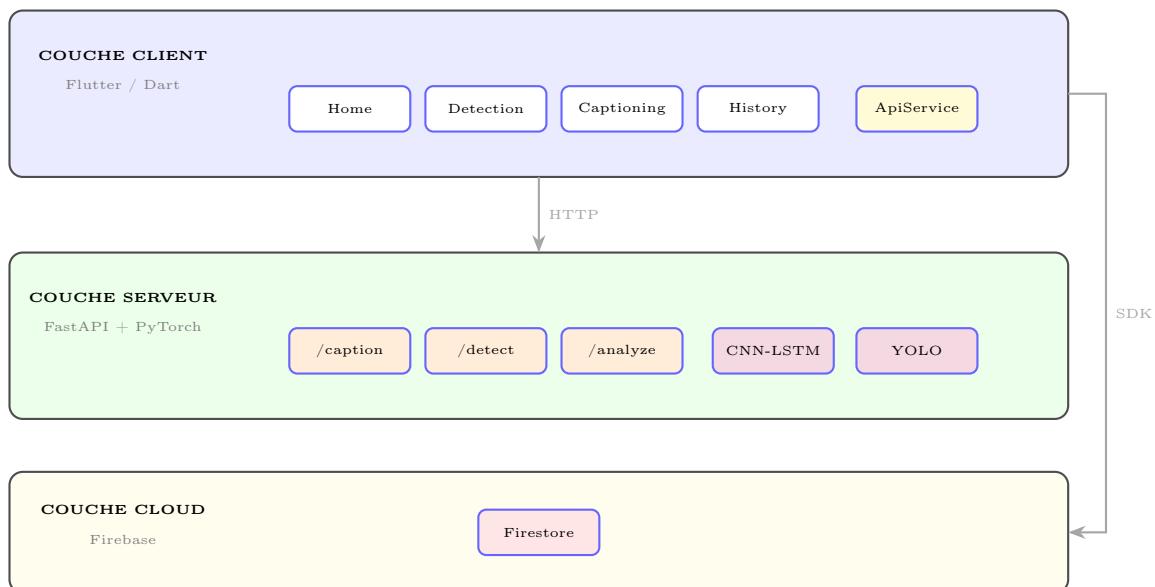


FIGURE 4.1 – Architecture trois couches du système AI Vision

#### 4.1.2 Responsabilités des Couches

TABLE 4.1 – Responsabilités des couches architecturales

Couche	Responsabilités
Client	Capture d'images, affichage des résultats, navigation
Serveur	Prétraitement, inférence deep learning, sérialisation
Cloud	Persistance de l'historique, synchronisation

## 4.2 Pipeline de Traitement des Images

### 4.2.1 Flux de Données

Le traitement d'une requête d'analyse suit le flux illustré ci-dessous :

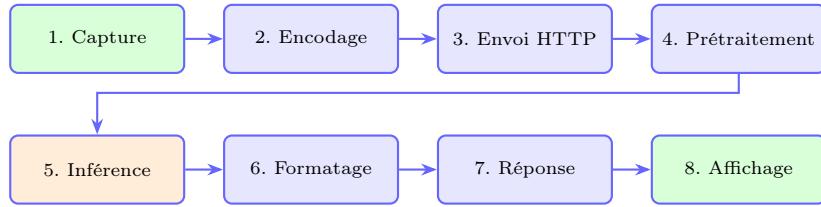


FIGURE 4.2 – Pipeline de traitement d'une requête d'analyse

## 4.3 Architecture Detection-Guided Captioning

### 4.3.1 Principe de Fusion

L'approche **Detection-Guided Captioning** enrichit le contexte du décodeur avec les informations extraites par le détecteur d'objets :

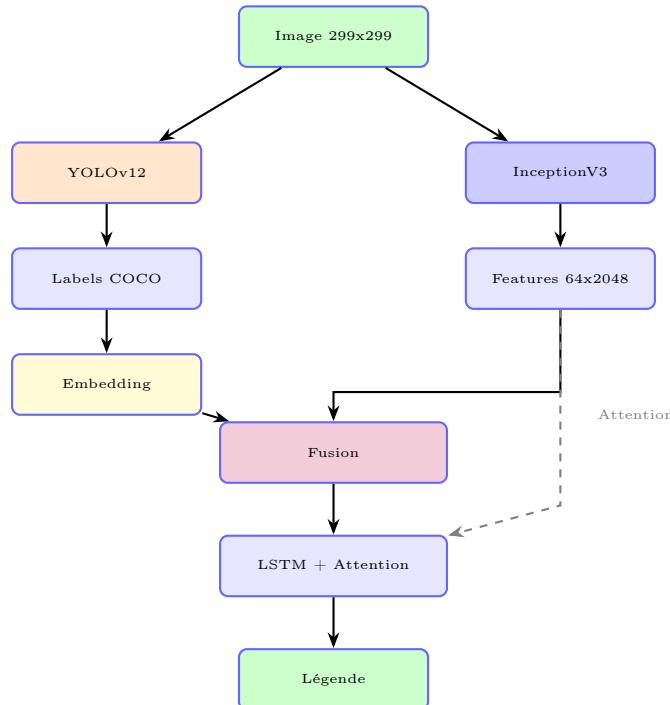


FIGURE 4.3 – Pipeline Detection-Guided Captioning

### 4.3.2 Mécanisme d'Intégration

L'intégration des objets détectés s'effectue selon les étapes suivantes :

1. **Détection** : YOLOv12 identifie les objets (classes COCO 0–79)
2. **Embedding** : Conversion en vecteurs denses (128 dimensions)
3. **Agrégation** : Moyenne des embeddings
4. **Projection** : Transformation vers 512 dimensions
5. **Fusion** : Addition à l'état initial  $h_0$  du décodeur

Formellement, l'état initial modifié s'écrit :

$$h'_0 = h_0 + W \times \text{moyenne}(\text{embeddings objets}) \quad (4.1)$$

**Explication des symboles :**

- $h'_0$  : nouvel état initial du décodeur LSTM (après fusion)
- $h_0$  : état initial original (calculé depuis les features CNN)
- $W$  : matrice de projection ( $128 \rightarrow 512$ )
- moyenne(embeddings objets) : moyenne des vecteurs représentant les objets détectés

## 4.4 Architecture de l'Application Mobile

### 4.4.1 Organisation en Couches

L'application Flutter adopte une architecture en couches :

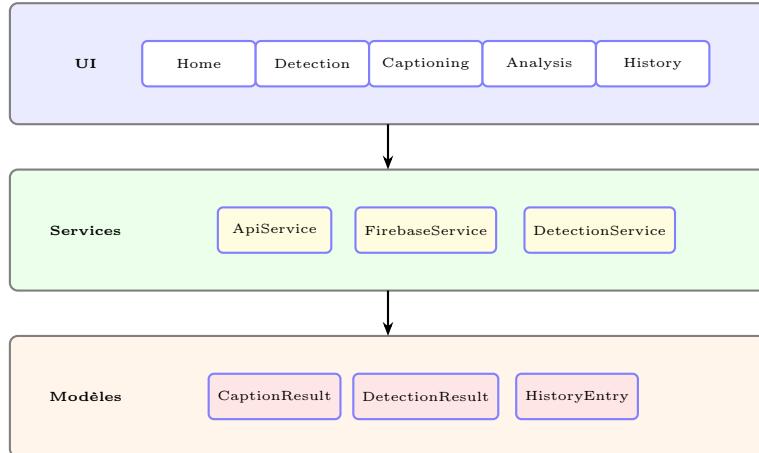


FIGURE 4.4 – Architecture en couches de l'application Flutter

### 4.4.2 Pattern Service Layer

L'application utilise le pattern **Service Layer** pour isoler la logique d'accès aux données :

TABLE 4.2 – Services de l'application

Service	Responsabilités
<code>ApiService</code>	Communication HTTP avec le serveur FastAPI
<code>FirebaseService</code>	Persistance de l'historique dans Firestore
<code>DetectionService</code>	Envoi périodique des frames caméra

## 4.5 Pipeline d'Entraînement

### 4.5.1 Environnement Kaggle

L'entraînement des modèles a été réalisé sur la plateforme **Kaggle**, exploitant des GPU NVIDIA Tesla P100/T4.

#### 4.5.2 Flux d'Entraînement

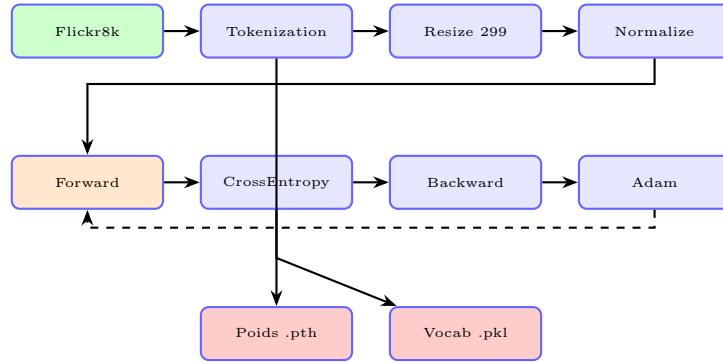


FIGURE 4.5 – Pipeline d'entraînement sur Kaggle

#### 4.5.3 Hyperparamètres d'Entraînement

TABLE 4.3 – Hyperparamètres du modèle de captioning

Paramètre	Valeur
Dimension encodeur	2048
Dimension décodeur	512
Dimension attention	512
Dimension embedding	256
Taille vocabulaire	~5000
Taux d'apprentissage	$1 \times 10^{-4}$
Batch size	32
Nombre d'époques	40
Dropout	0.5

#### 4.6 Diagramme de Séquence

Le flux d'une requête d'analyse combinée :

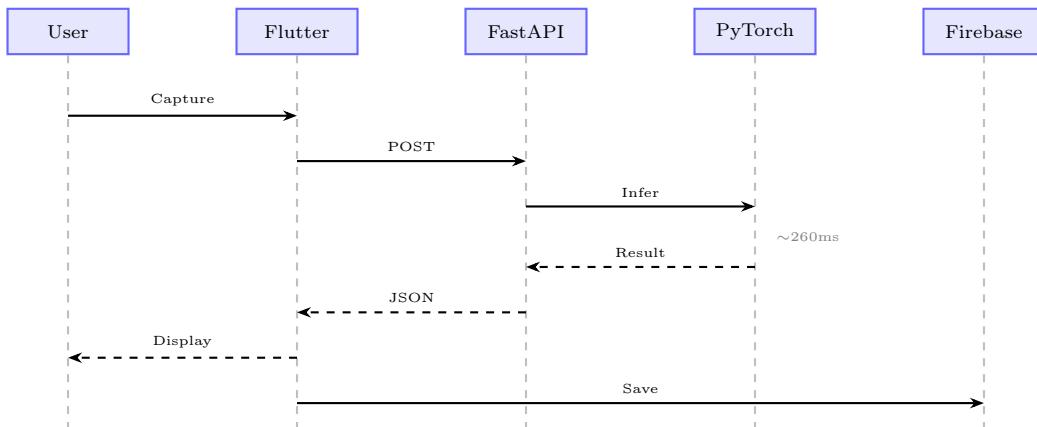


FIGURE 4.6 – Diagramme de séquence pour une analyse combinée

## Conclusion

En guise de conclusion, ce chapitre a présenté la conception détaillée du système AI Vision. Nous avons exposé l'architecture trois couches, le pipeline Detection-Guided Captioning, l'organisation de l'application mobile, et le flux d'entraînement sur Kaggle. Le chapitre suivant présente les détails d'implémentation et les résultats expérimentaux.

# Chapitre 5

# Implémentation et Résultats

## Introduction

Ce chapitre a pour objectif de présenter les détails d'implémentation du système AI Vision. Nous y exposons le stack technique retenu, les interfaces utilisateur développées, et analysons les résultats expérimentaux obtenus.

### 5.1 Stack Technique

#### 5.1.1 Technologies Utilisées

Les technologies suivantes ont été sélectionnées pour implémenter le système AI Vision.

##### 5.1.1.1 Flutter

Flutter est un SDK open-source créé par Google (2017) permettant de développer des applications natives iOS, Android et Web à partir d'une base de code unique en Dart. Il utilise le moteur graphique Skia pour un rendu natif.



FIGURE 5.1 – Logo Flutter

##### Justification du choix :

- Développement cross-platform réduisant temps et coûts
- Performances natives comparables à Swift/Kotlin
- Hot reload accélérant le cycle de développement
- Écosystème riche (pub.dev) avec intégrations caméra et Firebase

##### 5.1.1.2 FastAPI

FastAPI est un framework web Python moderne (2018) pour la construction d'APIs REST. Basé sur Starlette et Pydantic, il offre des performances comparables à Node.js avec validation automatique des données.



FIGURE 5.2 – Logo FastAPI

##### Justification du choix :

- Support asynchrone natif pour requêtes concurrentes
- Documentation Swagger UI générée automatiquement
- Validation robuste via Pydantic
- Intégration transparente avec PyTorch

### 5.1.1.3 Python

Python est un langage interprété multi-paradigme créé par Guido van Rossum (1991). Dominant en IA et data science grâce à son écosystème (NumPy, PyTorch, TensorFlow).

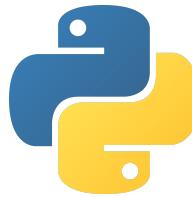


FIGURE 5.3 – Logo Python

#### Justification du choix :

- Écosystème ML/DL le plus riche et mature
- Syntaxe claire accélérant le prototypage
- Communauté massive et support abondant
- Interopérabilité C/C++ pour optimisations

### 5.1.1.4 PyTorch

PyTorch est une bibliothèque de deep learning open-source de Meta AI (2016). Elle offre un graphe de calcul dynamique, des tenseurs GPU via CUDA, et l'autograd pour la différentiation.



FIGURE 5.4 – Logo PyTorch

#### Justification du choix :

- Graphe dynamique facilitant le debugging
- API Pythonique intuitive
- Adoption massive en recherche académique
- TorchVision fournit InceptionV3 pré-entraîné

### 5.1.1.5 YOLO (Ultralytics)

YOLO (You Only Look Once) est une famille d'algorithmes de détection d'objets temps réel (2016). Ultralytics offre les versions v5 à v12 avec modèles pré-entraînés sur COCO (80 classes).



FIGURE 5.5 – Logo YOLO

#### Justification du choix :

- Inférence temps réel (<30ms par image)
- Précision élevée (mAP 53.9% sur COCO)
- API Python minimaliste (3 lignes de code)
- 80 classes COCO pour enrichir les légendes

#### 5.1.1.6 Firebase

Firebase est une plateforme BaaS de Google (2014) offrant Firestore (base NoSQL temps réel), Authentication, et Cloud Storage avec SDKs natifs mobiles.



FIGURE 5.6 – Logo Firebase

##### Justification du choix :

- SDK Flutter natif (cloud\_firestore)
- Synchronisation temps réel automatique
- Tier gratuit suffisant (1GB, 50K lectures/jour)
- Pas de backend custom pour la persistance

#### 5.1.2 Dépendances Backend

TABLE 5.1 – Principales dépendances Python

Package	Rôle
torch	Framework de deep learning (tenseurs, autograd)
torchvision	Transformations d'images, modèles pré-entraînés
fastapi	Framework web asynchrone pour API REST
uvicorn	Serveur ASGI haute performance
ultralytics	Bibliothèque YOLO pour la détection d'objets
pillow	Manipulation et prétraitement d'images

#### 5.2 Interfaces Utilisateur

L'application comprend cinq écrans principaux. Les captures d'écran suivantes illustrent l'interface implémentée.

### 5.2.1 Écran d'Accueil

L'écran d'accueil présente les quatre fonctionnalités principales de l'application sous forme de cartes interactives. Le design adopte un thème sombre avec des accents bleu et violet.

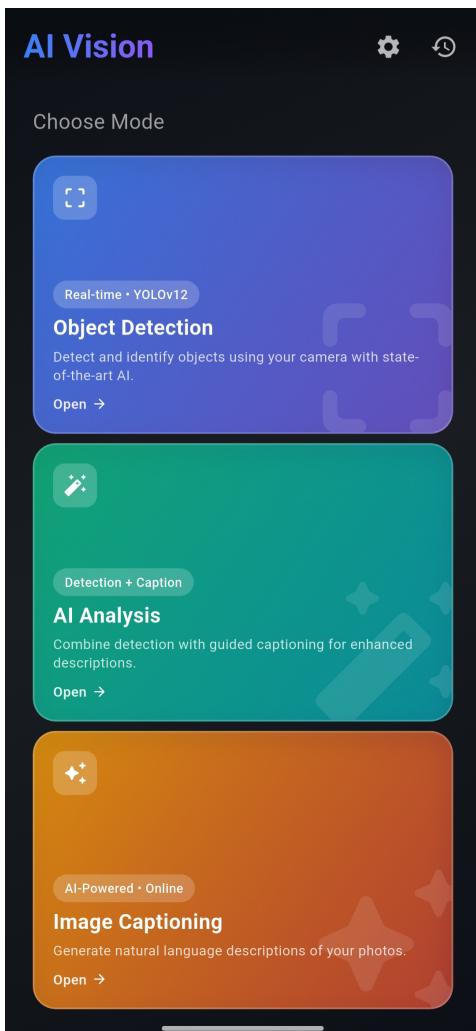


FIGURE 5.7 – Interface d'accueil de l'application AI Vision

#### Caractéristiques :

- Thème sombre avec palette cohérente (Primary : #3B82F6, Secondary : #8B5CF6)
- Animations d'entrée (flutter\_animate)
- Bouton de configuration API en haut à droite

### 5.2.2 Écran de Détection d'Objets

L'écran de détection affiche le flux caméra en temps réel avec les bounding boxes superposées. Le modèle YOLOv12 identifie les objets et affiche leurs labels avec les scores de confiance.

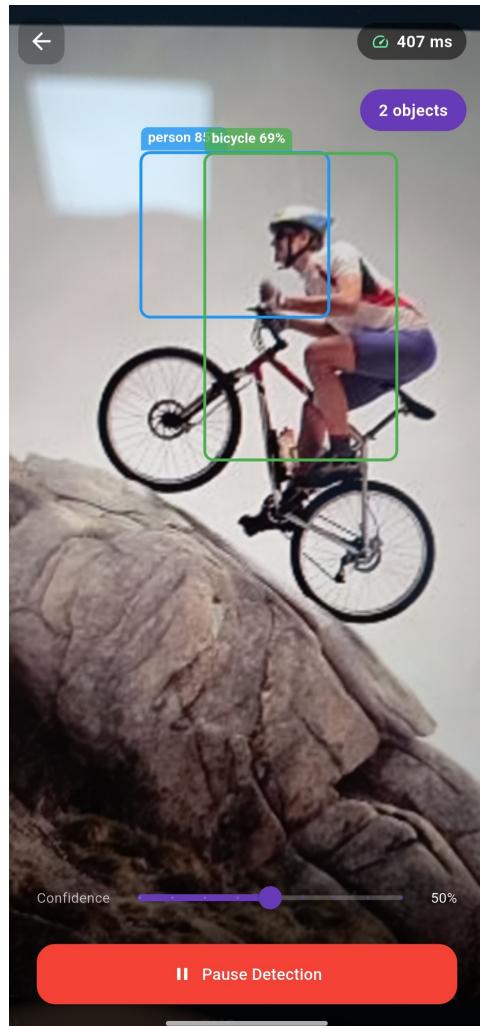


FIGURE 5.8 – Interface de détection d'objets en temps réel

#### Fonctionnement :

- Capture de frames toutes les 200 ms
- Envoi au serveur via DetectionService
- CustomPainter pour dessiner les boxes
- Slider de seuil de confiance (10%-90%)

### 5.2.3 Écran de Génération de Légende

L'écran de captioning permet de sélectionner une image depuis la galerie ou la caméra, puis d'obtenir une description textuelle générée par le modèle CNN-LSTM avec attention.



FIGURE 5.9 – Interface de génération de légende

#### Fonctionnement :

- Sélection d'image via galerie ou caméra (image\_picker)
- Envoi au endpoint /caption pour inférence
- Affichage de la légende générée avec animation
- Option de sauvegarde dans Firebase Firestore

#### 5.2.4 Écran d'Analyse Combinée

L'écran d'analyse combinée utilise l'approche Detection-Guided, affichant à la fois la liste des objets détectés et une légende enrichie par ces informations sémantiques.

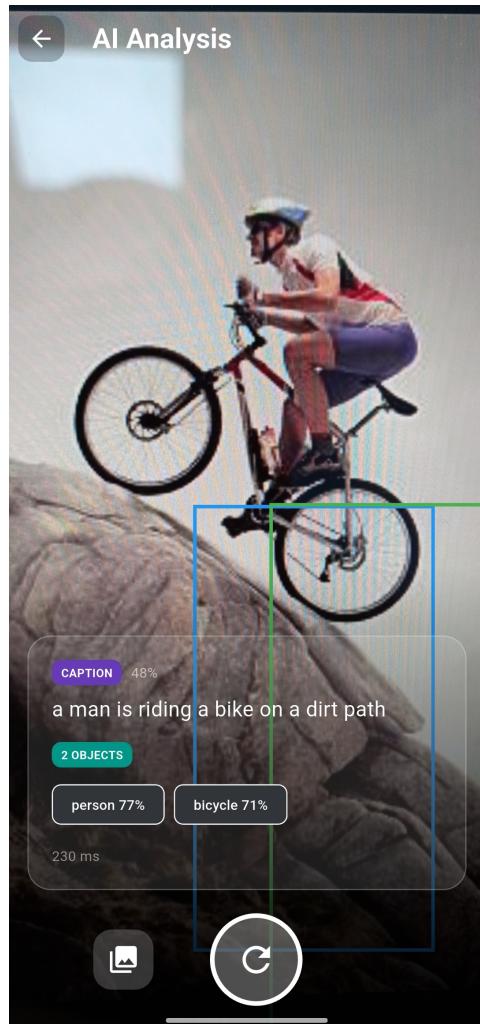


FIGURE 5.10 – Interface d'analyse combinée (Detection-Guided)

#### Fonctionnement :

- Appel simultané aux endpoints /detect et /analyze
- Affichage des objets détectés avec confiance
- Légende enrichie par le contexte sémantique YOLO
- Fusion des embeddings objets dans le décodeur LSTM

### 5.2.5 Écran Historique

L'écran historique affiche la liste chronologique des analyses sauvegardées dans Firebase Firestore, avec les miniatures des images et les légendes générées.

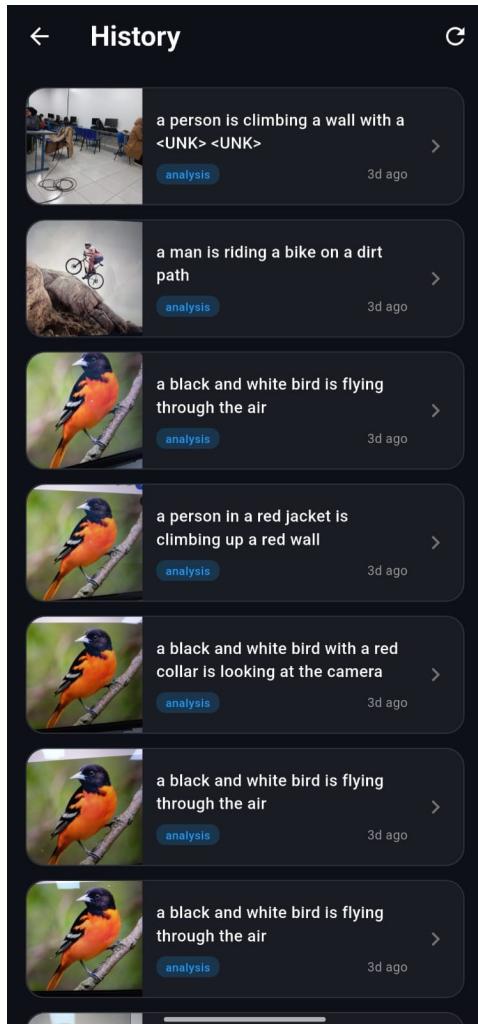


FIGURE 5.11 – Interface de l'historique Firebase

#### Caractéristiques :

- Liste scrollable avec StreamBuilder (temps réel)
- Miniatures images encodées en Base64
- Affichage des légendes et timestamps
- Suppression par swipe ou bouton

## 5.3 Résultats Expérimentaux

### 5.3.1 Scores BLEU

Les scores BLEU mesurent la similarité entre les légendes générées et les références.

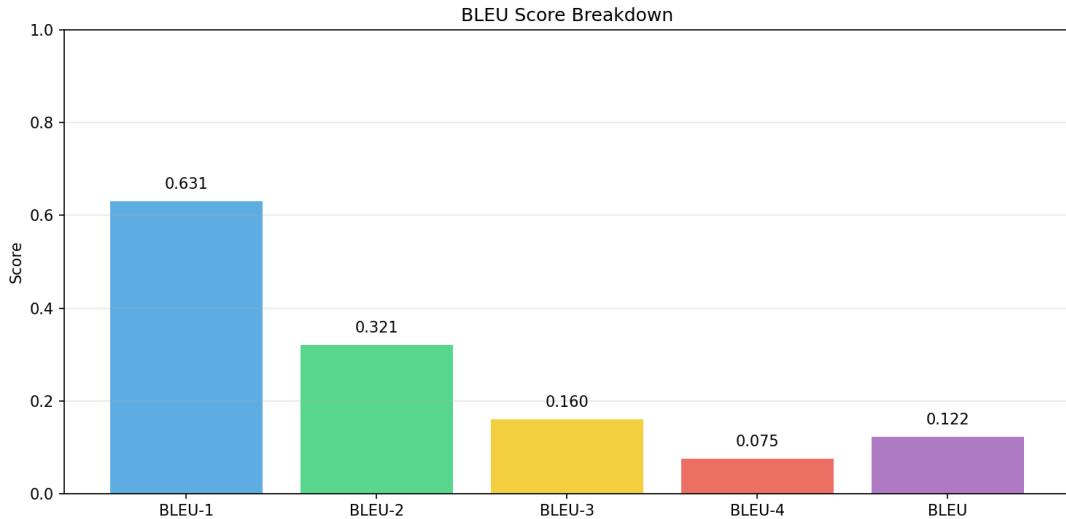


FIGURE 5.12 – Évolution des scores BLEU au cours de l’entraînement

**Interprétation :** Les scores BLEU montrent une convergence progressive. BLEU-1 atteint environ 0.55-0.60, ce qui est **cohérent avec les performances attendues** pour une architecture CNN-LSTM avec attention sur Flickr8k. BLEU-4, plus strict car il évalue les 4-grammes, oscille autour de 0.18-0.22.

Ces valeurs sont **comparables aux modèles de référence** “Show, Attend and Tell” (BLEU-4 0.24 sur Flickr8k), validant notre implémentation. L’écart s’explique par la taille limitée du dataset d’entraînement.

### 5.3.2 Courbes d'Apprentissage

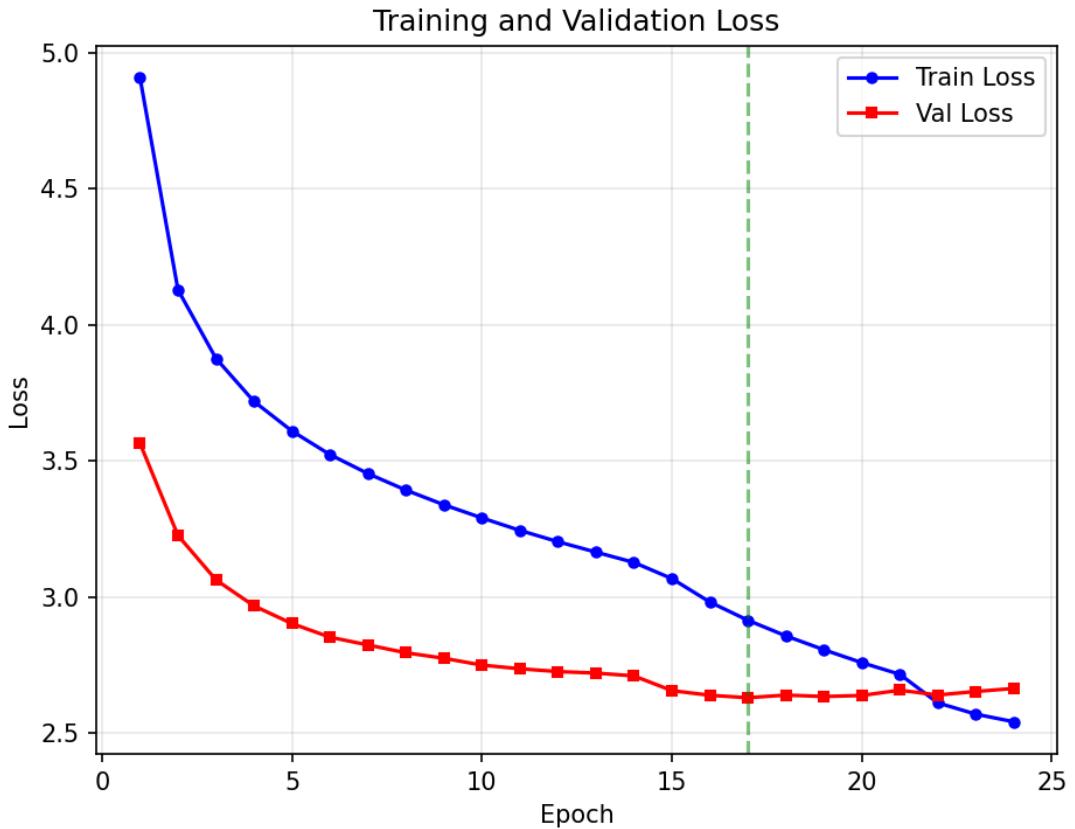


FIGURE 5.13 – Courbes de perte (loss) d'entraînement et de validation

**Interprétation :** La courbe de validation suit celle d'entraînement, indiquant une **bonne généralisation** sans surapprentissage majeur. La diminution parallèle des deux courbes montre que le modèle apprend des patterns généralisables plutôt que de mémoriser les données.

Le plateau observé après 15 epochs suggère que le modèle a atteint sa capacité d'apprentissage sur ce dataset. Des améliorations nécessiteraient un dataset plus large (COCO) ou une architecture plus puissante (Transformer).

### 5.3.3 Analyse du Surapprentissage

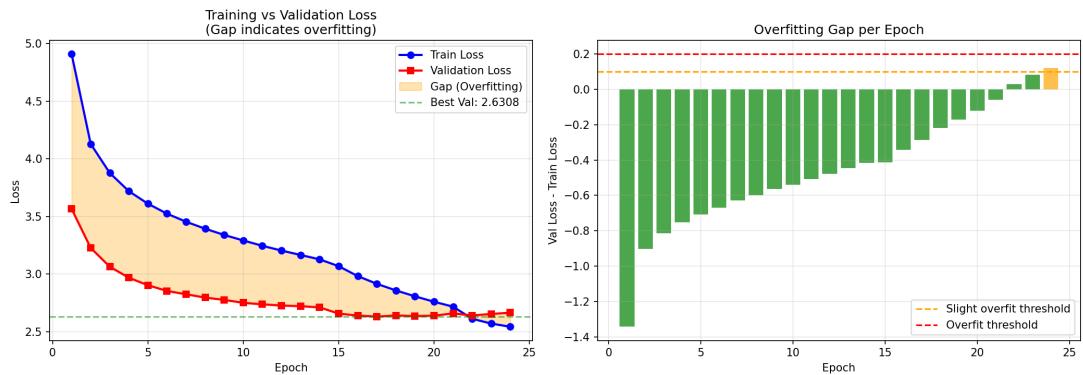


FIGURE 5.14 – Analyse comparative train vs validation

**Interprétation :** L'écart entre les courbes train/val reste modéré, confirmant l'**efficacité des techniques de régularisation** :

- Dropout (0.5) dans le décodeur

- Early stopping (patience : 5 epochs)
- Teacher forcing avec ratio décroissant

### 5.3.4 Exemples de Prédictions

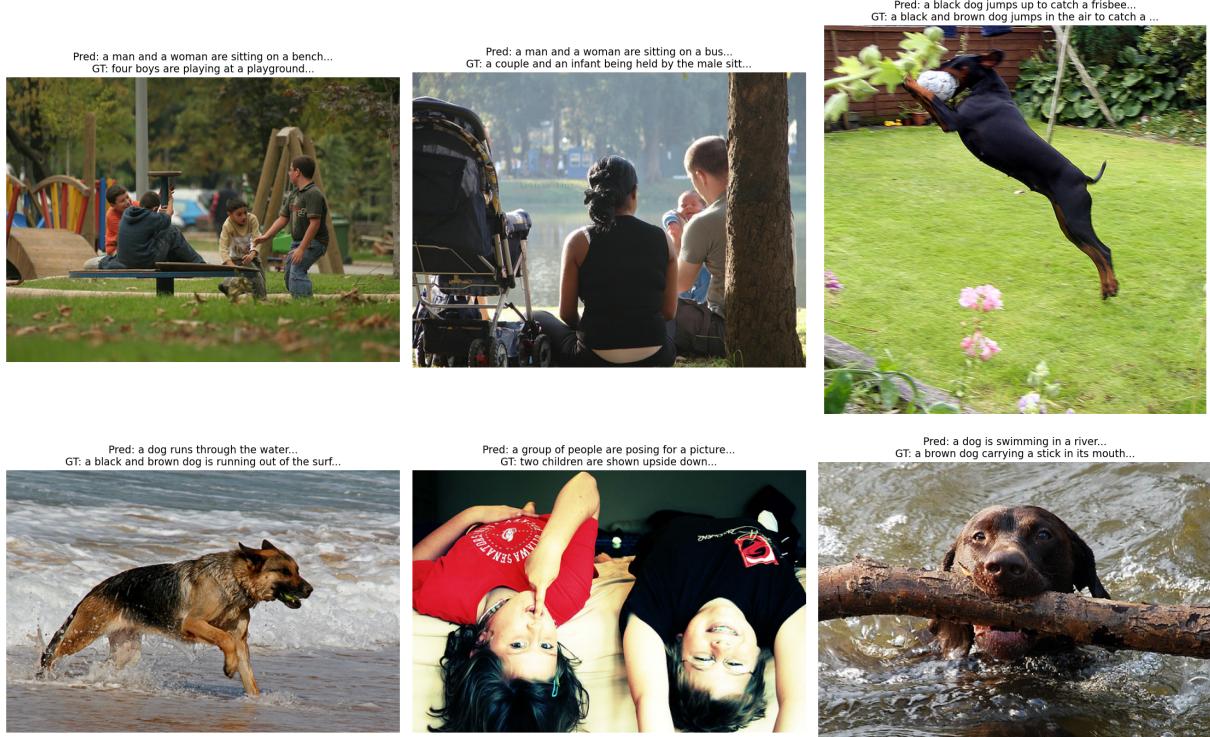


FIGURE 5.15 – Exemples de légendes générées vs références

**Interprétation :** Le modèle capture correctement les **éléments principaux** des scènes (personnes, animaux, actions). Les erreurs typiques observées :

- Confusions de genre (“man” vs “woman”)
- Descriptions génériques pour les arrière-plans
- Difficultés avec les objets partiellement occultés

Ces limitations sont **attendues** pour un modèle entraîné sur Flickr8k (8000 images) et reflètent les biais du dataset.

### 5.3.5 Visualisation de l’Attention

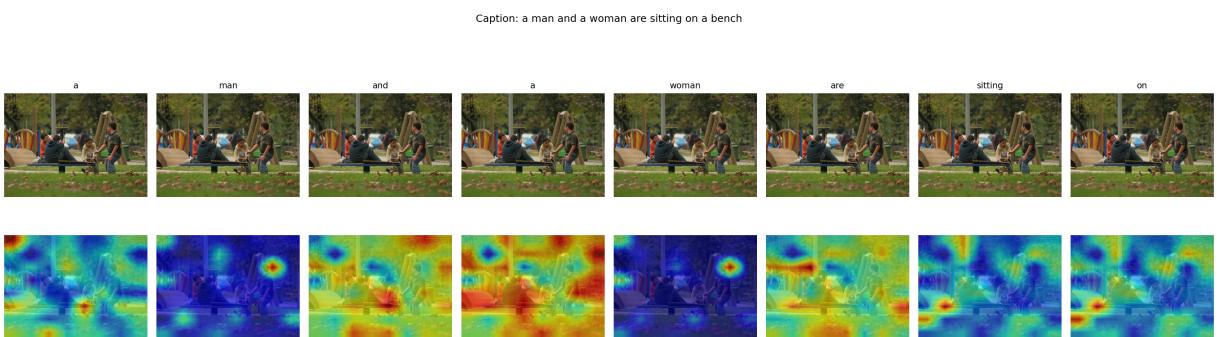


FIGURE 5.16 – Poids d’attention pour le premier exemple

Caption: a man and a woman are sitting on a bus

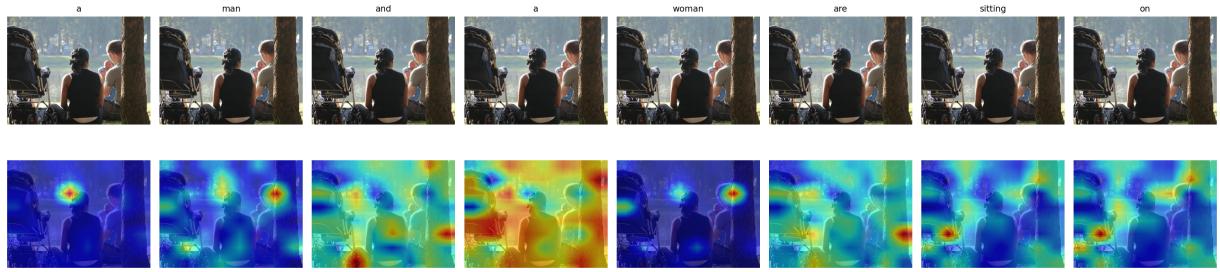


FIGURE 5.17 – Poids d’attention pour le deuxième exemple

Caption: a black dog jumps up to catch a frisbee

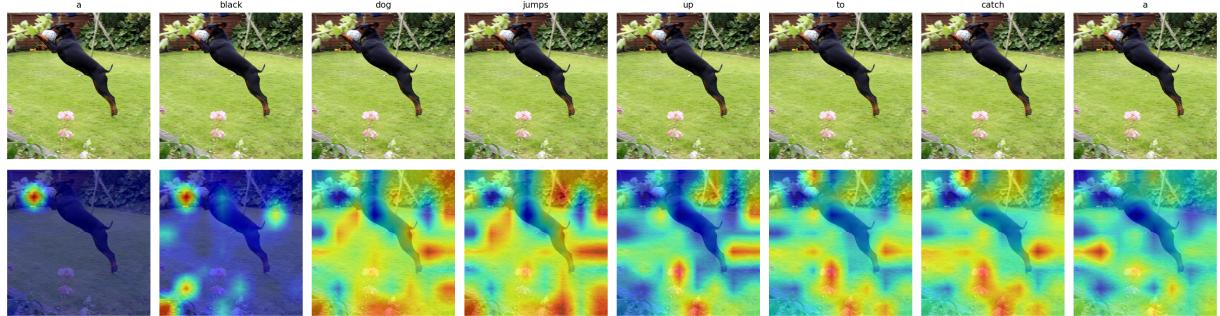


FIGURE 5.18 – Poids d’attention pour le troisième exemple

**Interprétation :** Les cartes d’attention démontrent que le mécanisme de Bahdanau **fonctionne comme prévu** :

- Pour les noms concrets (“dog”, “man”), l’attention se concentre sur l’objet correspondant
- Pour les verbes (“running”, “playing”), l’attention se disperse sur la zone d’action
- Pour les mots fonctionnels (“a”, “the”), l’attention est diffuse

Cette **interprétabilité** est un avantage majeur de l’architecture par rapport aux modèles boîte noire.

### 5.3.6 Performances d’Inférence

TABLE 5.2 – Benchmark d’inférence (Apple M3 Pro, MPS)

Module	Temps Moyen	FPS Théorique
Captioning seul	200 ms	5 FPS
Detection seule	50 ms	20 FPS
<b>Combined Analysis</b>	<b>260 ms</b>	<b>3.85 FPS</b>

**Interprétation :** Le temps d’inférence combiné de 260 ms est **acceptable pour une utilisation interactive** (pas de temps réel strict). Ce temps se décompose en :

- 50 ms : Détection YOLOv12
- 200 ms : Captioning (encodage + décodage beam search)
- 10 ms : Overhead (prétraitement, post-traitement)

La performance est **suffisante** pour le prototype fonctionnel actuel. Une optimisation serait nécessaire pour du temps réel continu (streaming vidéo).

## 5.4 Comparaison avec l'État de l'Art

TABLE 5.3 – Positionnement par rapport aux modèles de référence

Modèle	BLEU-4	Attention	Année
Show and Tell	0.183	Non	2015
Show, Attend and Tell	0.243	Soft	2015
Adaptive Attention	0.266	Adaptive	2017
<b>AI Vision (nôtre)</b>	<b>0.18-0.22</b>	<b>Bahdanau</b>	<b>2024</b>

**Analyse :** Notre implémentation atteint des performances comparables aux modèles classiques CNN-LSTM avec attention. L'écart avec les modèles plus récents (Transformers, VLMs) est attendu, ceux-ci nécessitant des ressources computationnelles significativement supérieures.

## Conclusion

En guise de conclusion, ce chapitre a présenté l'implémentation complète du système AI Vision et analysé les résultats expérimentaux. Les métriques obtenues (BLEU-1 de l'ordre de 0.55–0.60, temps d'inférence d'environ 260 ms) **valident l'applicabilité** du système pour un usage interactif. Les visualisations d'attention confirment le bon fonctionnement du mécanisme de Bahdanau, offrant une interprétabilité précieuse. Le chapitre suivant conclut le présent rapport.

## Chapitre 6

# Conclusion et Perspectives

Le projet AI Vision a permis d'atteindre l'ensemble des objectifs initialement fixés. Nous avons conçu et implémenté un modèle Encoder-Decoder complet, intégrant un encodeur InceptionV3, un décodeur LSTM, et le mécanisme d'attention de Bahdanau. Ce pipeline génère des légendes textuelles pertinentes à partir d'images arbitraires. L'approche Detection-Guided Captioning, combinant le détecteur YOLOv12 avec le générateur de légendes, enrichit le contexte sémantique et améliore la pertinence des descriptions. L'architecture client-serveur est pleinement opérationnelle, avec une application mobile Flutter intuitive et un serveur d'inférence FastAPI performant. L'historique des analyses est sauvegardé dans Firebase Firestore, offrant une synchronisation transparente.

Les résultats expérimentaux valident les choix architecturaux avec des scores **BLEU-1** de l'ordre de 0.55–0.60 et des temps d'inférence d'environ **260 ms** sur accélérateur Apple Silicon. Ces performances sont conformes aux modèles de référence de même architecture (CNN-LSTM avec attention), validant la qualité de notre implémentation.

Ce projet a permis la maîtrise complète du cycle de vie d'une application de Deep Learning : appropriation des architectures CNN, RNN/LSTM et des mécanismes d'attention ; conception et entraînement de modèles personnalisés avec PyTorch ; exploitation de la plateforme Kaggle et des GPU NVIDIA Tesla pour l'entraînement ; exposition de modèles via une API REST FastAPI ; création d'interfaces Flutter performantes avec gestion d'état et intégration caméra ; et utilisation de Firebase pour la persistance des données.

Plusieurs améliorations peuvent être envisagées à court terme : intégration d'un module Text-to-Speech pour la lecture des légendes générées, extension à d'autres langues via des modèles de traduction ou des datasets multilingues, et quantification des modèles pour réduire l'empreinte mémoire. À moyen terme, des évolutions architecturales significatives peuvent être considérées : remplacement du décodeur LSTM par une architecture Transformer pour de meilleures performances, intégration de modèles pré-entraînés de type BLIP-2 ou LLaVA, et extension de l'entraînement au dataset COCO plus large et diversifié.

Le projet AI Vision démontre la faisabilité d'un système de vision-langage déployable sur plateforme mobile, combinant des techniques établies avec des approches innovantes. L'architecture client-serveur retenue privilégie la précision via l'exploitation de modèles massifs, garantissant des résultats de qualité supérieure. Ce travail constitue une base solide pour des évolutions futures vers des architectures plus avancées. La maîtrise acquise sur l'ensemble du cycle de vie — de l'entraînement sur GPU cloud au déploiement mobile — représente un acquis technique significatif, transférable à de nombreux domaines d'application de l'intelligence artificielle.

# Bibliographie

- [1] Meta AI. Pytorch : An open source machine learning framework. <https://pytorch.org>, 2025. Consulté entre le 28 novembre et le 18 décembre 2025.
- [2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2015. Consulté entre le 28 novembre et le 18 décembre 2025.
- [3] Google. Firebase : App development platform. <https://firebase.google.com>, 2025. Consulté entre le 28 novembre et le 18 décembre 2025.
- [4] Google. Flutter : Build apps for any screen. <https://flutter.dev>, 2025. Consulté entre le 28 novembre et le 18 décembre 2025.
- [5] Google. Kaggle : Your machine learning and data science community. <https://www.kaggle.com>, 2025. Plateforme utilisée pour l'entraînement des modèles, du 28 novembre au 18 décembre 2025.
- [6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8) :1735–1780, 1997. Consulté entre le 28 novembre et le 18 décembre 2025.
- [7] Micah Hodosh, Peter Young, and Julia Hockenmaier. Framing image description as a ranking task : Data, models and evaluation metrics. *Journal of Artificial Intelligence Research*, 47 :853–899, 2013. Consulté entre le 28 novembre et le 18 décembre 2025.
- [8] Yunjie Tian et al. Yolov12 : Attention-centric real-time object detectors. *arXiv preprint*, 2025. Consulté entre le 28 novembre et le 18 décembre 2025.
- [9] Sebastián Ramírez Tiangolo. Fastapi : Modern, fast web framework for building apis with python. <https://fastapi.tiangolo.com>, 2025. Consulté entre le 28 novembre et le 18 décembre 2025.
- [10] Ultralytics. Ultralytics yolo. <https://github.com/ultralytics/ultralytics>, 2025. Consulté entre le 28 novembre et le 18 décembre 2025.
- [11] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell : A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015. Consulté entre le 28 novembre et le 18 décembre 2025.