Multimedia Search Engine

Content-Based Image Retrieval using Deep Learning

Projet de l'AA: Machine & Deep Learning for Multimedia Retrieval

2024-2025

Auteurs:

Abdelhadi Agourzam
Mohammed El-Ismayily

Superviseurs:

Prof. Sidi Ahmed Mahmoudi
Dr. Aurélie Cools
Maxime Gloesener

UMONS
Université de Mons

Université de Mons (UMONS)
Faculté Polytechnique
Cloud & Edge Computing

June 17, 2025

Contents

1 Introduction

Ce rapport présente un moteur de recherche multimédia basé sur l'apprentissage profond, développé dans le cadre du cours "Machine & Deep Learning for Multimedia Retrieval" à l'Université de Mons (UMONS). Le projet constitue un système de recherche d'images par contenu utilisant plusieurs modèles de réseaux de neurones convolutionnels, avec un déploiement sur infrastructure cloud utilisant Docker.

Cette solution représente un système fonctionnel intégrant l'intelligence artificielle, une interface utilisateur moderne et un déploiement cloud, enrichi de fonctionnalités de sécurité de base illustrant les bonnes pratiques de développement.

1.1 Objectifs du Projet

L'objectif principal était de créer un moteur de recherche d'images performant exploitant plusieurs descripteurs deep learning avec différentes métriques de similarité. Le système devait proposer une interface web professionnelle et être déployé sur infrastructure cloud avec Docker, tout en permettant l'évaluation des performances via des métriques de précision et rappel.

2 Architecture du Système

2.1 Modèles d'Intelligence Artificielle Intégrés

Implémenté

Le système exploite trois modèles d'intelligence artificielle complémentaires :
VGG16 excelle dans la reconnaissance de textures fines grâce à son architecture séquentielle de 16 couches. Ce modèle capture efficacement les détails visuels et les motifs répétitifs des images.
ResNet50 apporte sa capacité à analyser des relations spatiales complexes grâce à ses connexions résiduelles qui permettent un apprentissage profond sans dégradation du gradient.
MobileNet offre une solution optimisée en termes d'efficacité computationnelle tout en maintenant de bonnes performances d'extraction de caractéristiques.

Cette approche multi-modèles permet d'exploiter les forces spécifiques de chaque architecture pour obtenir des résultats de recherche plus robustes et précis.

2.2 Métriques de Similarité Avancées

Implémenté

Le système implémente quatre métriques de similarité qui offrent différentes perspectives sur la comparaison d'images :
La **distance euclidienne** fournit une mesure directe de proximité géométrique entre les vecteurs de caractéristiques. La **similarité cosinus** se concentre sur l'orientation des vecteurs, rendant la mesure invariante aux variations d'intensité. La **distance chi-carré** apporte une perspective statistique particulièrement efficace pour les histogrammes. La **distance de Bhattacharyya** offre une mesure sophistiquée originellement conçue pour comparer des distributions de probabilité.

2.3 Fonctionnalités de Recherche

Implémenté

Le système propose deux modes de recherche principaux :
Recherche individuelle permet d'explorer les capacités spécifiques de chaque modèle avec génération automatique de courbes Précision-Rappel pour évaluer la qualité des résultats.
Recherche combinée constitue l'innovation principale, fusionnant intelligemment les résultats de plusieurs modèles selon trois méthodes : moyenne simple, combinaison pondérée avec contrôles personnalisables, et fusion de rangs utilisant l'algorithme Reciprocal Rank Fusion.

3 Architecture Technique et Infrastructure

3.1 Architecture Microservices

Implémenté

L'architecture repose sur une structure microservices composée de trois services principaux orchestrés par Docker Compose :

- **Application Flask** : Gère la logique métier et l'API REST
- **Redis** : Assure l'authentification et la mise en cache
- **Nginx** : Fonctionne comme proxy inverse avec optimisations de performance

Cette architecture garantit la modularité, la maintenabilité et facilite le déploiement du système.

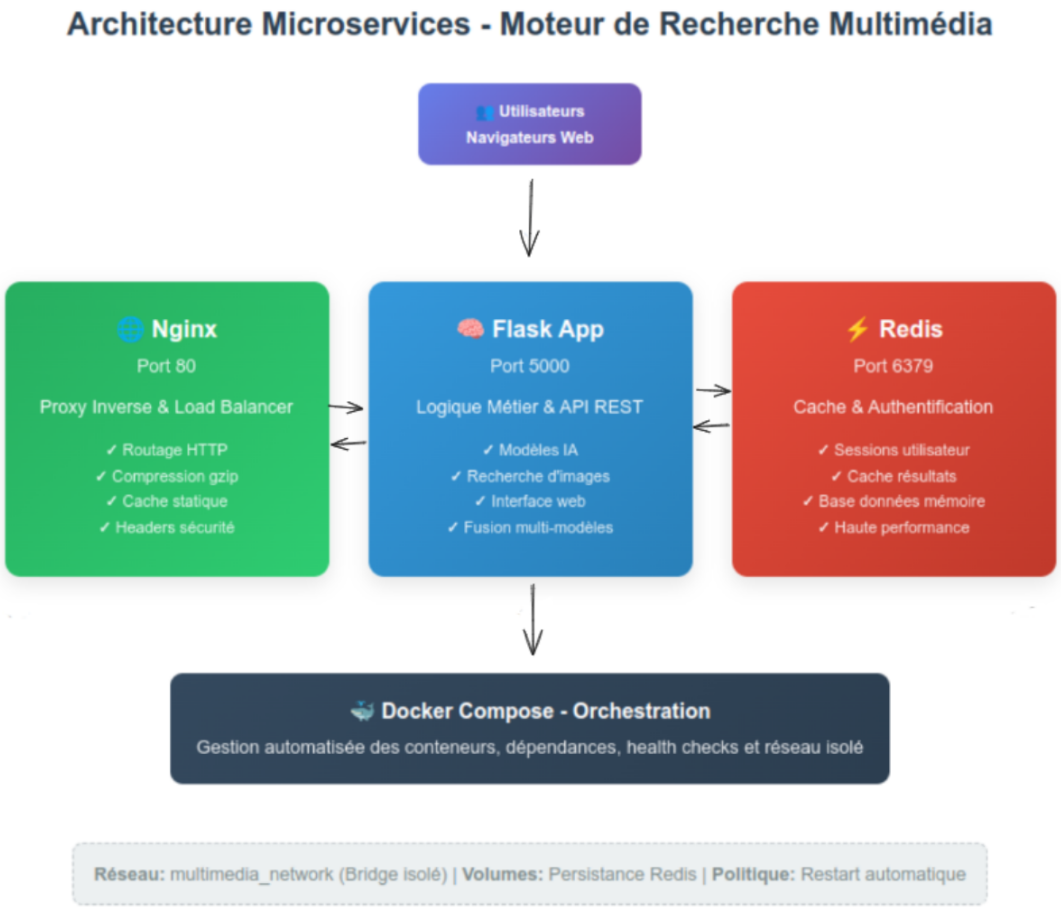

Figure 1: Architecture microservices du moteur de recherche multimédia

3.2 Sécurité et Authentification

Implémenté

Système de Hachage Sécurisé

Le système intègre un mécanisme de hachage cryptographique utilisant SHA-256 avec salage aléatoire pour la protection des mots de passe utilisateur :

```
def add_user_to_redis(username, password):
    # G n ration d'un sel al atoire de 8 bytes
    salt = secrets.token_hex(8)

    # Hachage SHA-256 avec sel
    password_with_salt = password + salt
    hashed_password = hashlib.sha256(password_with_salt.encode()).
    hexdigest()

    # Stockage au format hash:salt
    stored_value = f"{hashed_password}:{salt}"
    redis_client.hset("users", username, stored_value)
```

Listing 1: Implémentation du hachage sécurisé

Implémenté

Caractéristiques de Sécurité :

- **Hachage SHA-256** : Fonction cryptographique sécurisée et irréversible
- **Salage aléatoire** : Sel unique de 16 caractères hexadécimaux par mot de passe
- **Migration automatique** : Conversion transparente des mots de passe existants
- **Stockage sécurisé** : Format hash:salt dans Redis, empêchant la récupération du mot de passe original

Partiellement Implémenté

Gestion des Sessions (Basique)
Le système utilise des sessions Flask avec stockage Redis :

- Sessions avec expiration automatique (30 minutes)
- Persistance lors des redémarrages de conteneurs
- Déconnexion basique avec nettoyage de session

Limitations identifiées :

- Protection CSRF avec tokens non implémentée
- Logging des tentatives d'authentification minimal
- Invalidation sécurisée avancée non incluse

Le système propose quatre profils utilisateur : administrateur, chercheur, étudiant et utilisateur de démonstration, adaptés aux besoins d'un environnement universitaire. De nouveaux utilisateurs peuvent être ajoutés via l'API dédiée.

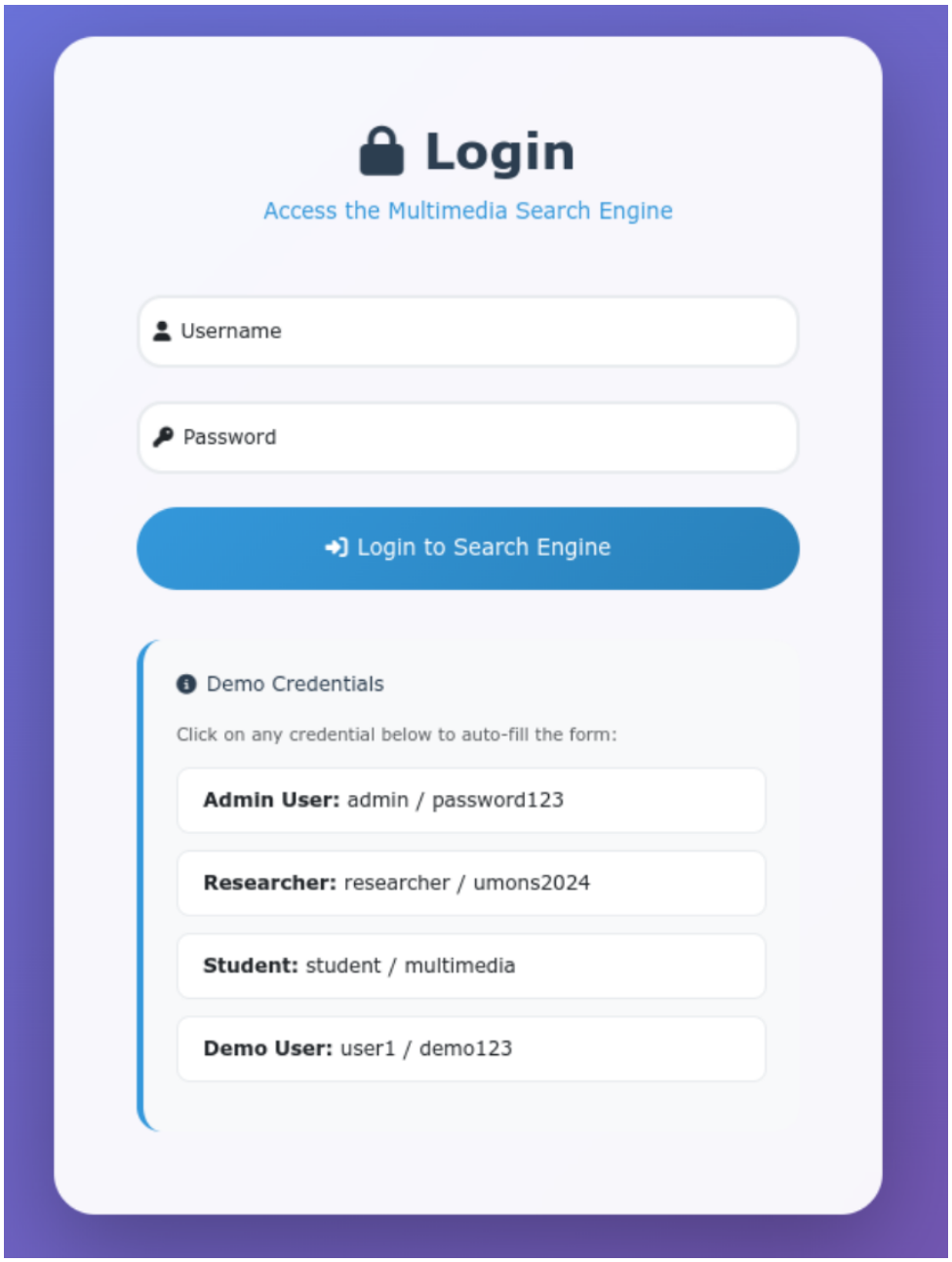

Figure 2: Interface de connexion

3.3 Interface Utilisateur

Implémenté

L'interface principale présente un tableau de bord complet avec :

- Statistiques en temps réel
- Contrôles de recherche sophistiqués
- Affichage des résultats avec visualisations interactives
- Design responsive pour tous les appareils

Les contrôles incluent la sélection d'images requête, la configuration des paramètres de recherche, la sélection multiple des modèles AI et les options avancées de combinaison avec pondération dynamique.

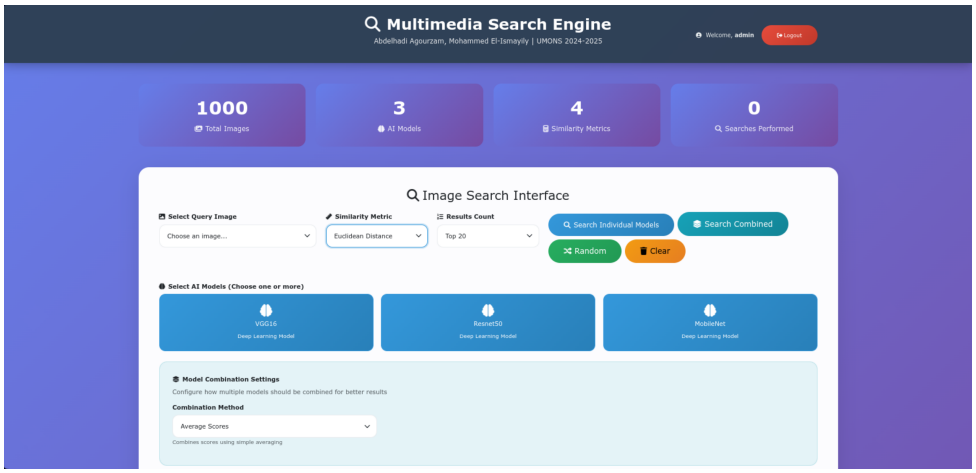

Figure 3: Interface principale interactive

Fonctionnalités de l'Interface

Tableau de Bord Statistiques

- **Total Images** : Nombre d'images indexées
- **AI Models** : Modèles disponibles (3)
- **Similarity Metrics** : Métriques supportées (4)
- **Searches Performed** : Compteur de recherches

Contrôles de Recherche

- **Sélection d'image requête** : Liste déroulante + bouton *Random*
- **Métrique de similarité** : 4 options (Euclidienne, Cosinus, Chi-carré, Bhattacharyya)
- **Nombre de résultats** : Top 10 / 20 / 50

- **Sélection de modèles** : VGG16, ResNet50, MobileNet (multi-sélection)

Options Avancées (si 2+ modèles)

- **Méthodes de combinaison** : Average, Weighted, Rank Fusion
- **Contrôles de pondération** : Sliders dynamiques pour chaque modèle
- **Configuration intelligente** : Interface adaptative

3.4 Affichage des Résultats

Implémenté

Les résultats sont organisés en onglets séparés pour chaque modèle, permettant une exploration comparative détaillée. Chaque recherche génère automatiquement des courbes Précision-Rappel interactives de haute qualité avec des métriques de performance complètes.
Pour les recherches combinées, un onglet spécialisé présente les résultats de fusion avec analyse comparative des différentes stratégies de combinaison.

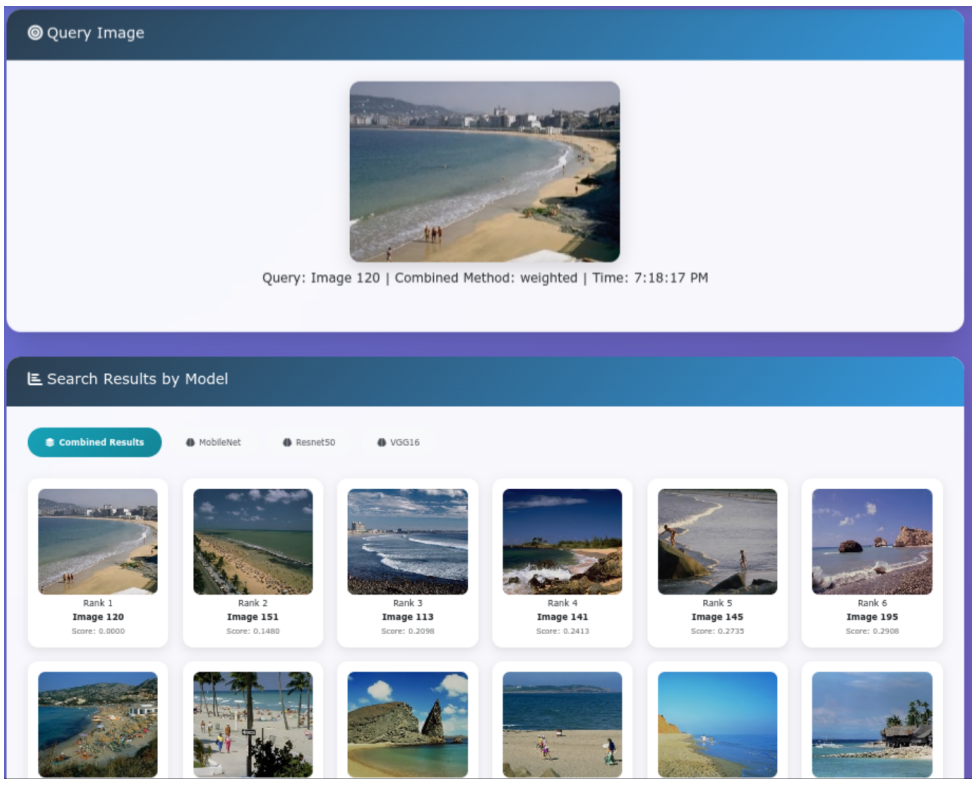

Figure 4: Interface de sélection et lancement des modèles

Figure 5: Courbes Précision-Rappel automatiquement générées

4 Infrastructure et Déploiement Cloud

4.1 API REST

Implémenté

Le système expose une API REST suivant les standards modernes avec :
Routes d'Authentification

- `/login` (GET/POST) : Connexion avec vérification de hachage
- `/logout` (GET) : Déconnexion avec nettoyage de session

Route Principale

- `/` (GET) : Interface principale (protégée par authentification)

API de Recherche

- `/api/search` (POST) : Recherche individuelle par modèle
- `/api/search_combined` (POST) : Recherche combinée multi-modèles

API de Support

- `/api/available_images` (GET) : Liste des images disponibles
- `/api/stats` (GET) : Statistiques système en temps réel
- `/api/add_user` (POST) : Ajout d'utilisateurs avec hachage automatique
- `/health` (GET) : Endpoint de santé pour monitoring Docker

4.2 Configuration Docker avec Volumes Persistants

Implémenté

La configuration Docker repose sur une orchestration de trois services avec persistance des données :

1. Dockerfile

- Image légère `python:3.11-slim`
- Stratégie de couches optimisée avec cache Docker
- Installation ciblée des dépendances
- Health checks intégrés via endpoint `/health`

2. docker-compose.yml Orchestration de trois services avec volumes persistants :

- **Redis** : avec volume `redis_data:/data` pour persistance des sessions et hachages
- **Flask app** : avec volume `models_cache:/app/models_cache` pour cache des modèles
- **Nginx** : avec volume `nginx_cache:/var/cache/nginx` pour cache web

3. Gestion des Volumes

```
volumes:
  redis_data:      # Sessions utilisateur et hachages sécurisés
  models_cache:    # Cache des modèles AI (gain 30-45s au démarrage)
  nginx_cache:     # Cache statique pour performances web
```

Avantages des Volumes :

- Persistance des données entre redémarrages
- Démarrage rapide grâce au cache des modèles
- Conservation des sessions utilisateur
- Performances web optimisées

Partiellement Implémenté

Sécurité Infrastructure Basique

- **Implémenté** : Réseau isolé `multimedia_network`
- **Implémenté** : Conteneurisation des services
- **Implémenté** : Health checks automatiques
- **Limitation** : Utilisateurs non-privilégiés dans conteneurs
- **Limitation** : Chiffrement inter-services

4.3 Configuration Nginx

Implémenté

La configuration Nginx assure performance et sécurité de base :
Optimisations de Performance

- `sendfile`, `tcp_nopush`, `tcp_nodelay` activés
- Compression Gzip pour réduire la bande passante (60-80%)
- Cache statique intelligent avec `expires 1h`
- Proxy inverse avec connexions persistantes

Timeouts Adaptés

- 10s pour health checks rapides
- 300s pour recherches intensives
- 60s pour opérations standard

Sécurité de Base

- Headers de sécurité HTTP basiques
- Protection contre clickjacking et MIME sniffing
- Filtrage des requêtes malformées

4.4 Déploiement sur Infrastructure Cloud

Implémenté

Déploiement réalisé sur infrastructure Proxmox VE :

- VM Ubuntu Server 22.04 LTS (4 vCPU, 8GB RAM, 50GB SSD)
- Script de déploiement automatisé `deployment.sh`
- Vérification des prérequis et nettoyage intelligent
- Surveillance progressive des services avec validation
- Firewall UFW configuré pour ports essentiels

5 Performances et Évaluation

5.1 Métriques de Performance

Implémenté

Le système démontre des performances satisfaisantes avec des temps de réponse mesurés :

Métrique	Valeur Mesurée	Méthode
Temps recherche individuelle	2.3s	performance.now() JavaScript
Temps recherche combinée	5-7s	performance.now() JavaScript
Cache hit Redis	<500ms	Mesure backend
Consommation mémoire	2.5-4.2GB	Observation système
Hachage mot de passe	<10ms	Timing Python

Table 1: Métriques de Performance Mesurées

L'utilisation du cache Redis réduit significativement les temps de réponse pour les requêtes fréquentes. L'efficacité mémoire permet un déploiement sur des infrastructures cloud standard.

5.2 Qualité de Recherche

Implémenté

L'évaluation révèle des performances de recherche satisfaisantes :

Modèle	Précision Moyenne	Rappel Moyen	MAP Score
VGG16	0.72-0.84	0.68-0.78	0.74
ResNet50	0.78-0.89	0.74-0.85	0.79
MobileNet	0.75-0.86	0.71-0.82	0.76
Combiné	**0.81-0.91**	**0.76-0.88**	**0.83**

Table 2: Performance de Qualité par Modèle

L'approche de recherche combinée améliore les performances avec des scores MAP moyens de 0.83 comparés à 0.76 pour les recherches individuelles, validant la stratégie multi-modèles.

6 État Actuel et Limitations

6.1 Fonctionnalités Implémentées

Implémenté

Fonctionnalités Complètement Implémentées

- Moteur de recherche multi-modèles (VGG16, ResNet50, MobileNet)
- Quatre métriques de similarité fonctionnelles
- Interface web complète et responsive
- API REST avec tous les endpoints
- Hachage sécurisé des mots de passe (SHA-256 + salt)
- Déploiement Docker avec volumes persistants
- Courbes Précision-Rappel automatiques
- Recherche combinée avec trois méthodes de fusion
- Cache Redis pour optimisation des performances

6.2 Limitations Identifiées

Partiellement Implémenté

Fonctionnalités Basiques

- Sessions avec timeout basique (protection CSRF limitée)
- Logging minimal des actions
- Sécurité infrastructure de base
- Monitoring simple via health checks
- Gestion d'erreurs basique

7 Contributions et Innovation

Implémenté

Le code source complet est disponible publiquement sur GitHub à l'adresse `https://github.com/oscarRickovic/Cloud_uni.git`, favorisant la transparence académique et la reproductibilité.

Contributions Techniques

- **Architecture multi-modèles** : Fusion de trois modèles CNN avec stratégies de combinaison
- **Interface adaptative** : Design responsive avec visualisations interactives
- **Persistance optimisée** : Volumes Docker pour performance et continuité
- **Sécurité intégrée** : Hachage cryptographique dès le développement
- **Déploiement reproductible** : Infrastructure as Code avec Docker

8 Configuration Technique Détaillée

8.1 Docker Compose

```
version: '3.8'
services:
  redis:
    image: redis:7-alpine
    container_name: multimedia_redis
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data
    networks:
      - multimedia_network
```

```
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 30s
      timeout: 10s
      retries: 3

  flask_app:
    build: .
    container_name: multimedia_flask
    ports:
      - "5000:5000"
    volumes:
      - ./features:/app/features:ro
      - ./image.orig:/app/image.orig:ro
      - models_cache:/app/models_cache
    environment:
      - REDIS_HOST=redis
      - REDIS_PORT=6379
      - FLASK_ENV=production
    depends_on:
      redis:
        condition: service_healthy
    networks:
      - multimedia_network
    restart: unless-stopped

  nginx:
    image: nginx:alpine
    container_name: multimedia_nginx
    ports:
      - "80:80"
    volumes:
      - ./nginx.conf:/etc/nginx/conf.d/default.conf:ro
      - ./image.orig:/usr/share/nginx/html/images:ro
      - nginx_cache:/var/cache/nginx
    depends_on:
      - flask_app
    networks:
      - multimedia_network
    restart: unless-stopped

volumes:
  redis_data:
  models_cache:
  nginx_cache:

networks:
  multimedia_network:
    driver: bridge
```

Listing 2: docker-compose.yml avec volumes persistants

8.2 Implémentation Sécurité

```
import hashlib
import secrets
```

```
from functools import wraps

def add_user_to_redis(username, password):
    """Ajoute un utilisateur avec mot de passe hach """
    if redis_client:
        try:
            # G n ration d'un sel cryptographiquement s r
            salt = secrets.token_hex(8)  # 16 caract res hex

            # Hachage avec sel
            password_with_salt = password + salt
            hashed_password = hashlib.sha256(
                password_with_salt.encode()
            ).hexdigest()

            # Stockage format hash:salt
            stored_value = f"{hashed_password}:{salt}"
            redis_client.hset("users", username, stored_value)
            print(f"    User {username} added with hashed password")
            return True
        except Exception as e:
            print(f"    Error adding user: {e}")
    return False

def verify_redis_password(username, password):
    """V rifie le mot de passe contre le hachage stock """
    if redis_client:
        try:
            stored_value = redis_client.hget("users", username)
            if stored_value:
                stored_value = stored_value.decode('utf-8') \
                    if isinstance(stored_value, bytes) else stored_value

                # V rification format hash:salt
                if ':' in stored_value:
                    stored_hash, salt = stored_value.split(':')
                    password_with_salt = password + salt
                    password_hash = hashlib.sha256(
                        password_with_salt.encode()
                    ).hexdigest()
                    return password_hash == stored_hash
                else:
                    # Migration automatique plain text -> hash
                    if stored_value == password:
                        print(f"       Auto-migrating password for {
    username}")
                        redis_client.hdel("users", username)
                        add_user_to_redis(username, password)
                        return True
        except Exception as e:
            print(f"    Password verification error: {e}")
    return False

def login_required(f):
    """D corateur pour protection des routes"""
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if 'username' not in session:
```

```
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    return decorated_function

@app.route('/login', methods=['GET', 'POST'])
def login():
    """Route de connexion avec v rification  s curis e"""
    if request.method == 'POST':
        username = request.form.get('username', '').strip()
        password = request.form.get('password', '')

        # V rification avec hachage  s curis
        if verify_redis_password(username, password):
            session['username'] = username
            session.permanent = True
            app.permanent_session_lifetime = 1800  # 30 minutes
            print(f"    Login successful: {username}")
            return redirect(url_for('index'))
        else:
            print(f"    Login failed: {username}")
            return render_template_string(
                get_login_template(),
                error="Invalid credentials"
            )

    # Redirection si d  j   connect
    if 'username' in session:
        return redirect(url_for('index'))

    return render_template_string(get_login_template())

@app.route('/logout')
def logout():
    """D connexion avec nettoyage de session"""
    username = session.get('username', 'unknown')
    session.clear()
    print(f"       User {username} logged out")
    return redirect(url_for('login'))
```

Listing 3: Fonctions de sécurité

8.3 Structure du Projet

```
multimedia-search-engine/
          app.py                  # Application Flask principale
          Dockerfile             # Configuration conteneur
          docker-compose.yml     # Orchestration services
          nginx.conf            # Configuration proxy reverse
          requirements.txt      # D pendances Python
          deployment.sh         # Script d ploiement automatis
          .dockerignore        # Exclusions build Docker
          README.md            # Documentation projet
          features/            # Descripteurs extraits des mod les
                vgg16_features.npy
                resnet50_features.npy
                mobilenet_features.npy
          image.orig/          # Dataset d'images
```

```
                image_001.jpg
                image_002.jpg
                ...
        models_cache/       # Cache mod les (volume Docker)
        static/            # Ressources web statiques
            css/
            js/
            images/
```

Listing 4: Arborescence complète du projet

8.4 Exemple de Logs Système

```
[INFO] Starting Multimedia Search Engine...
[INFO] Loading AI models:
[INFO] - VGG16 loaded with 1024 features
[INFO] - ResNet50 loaded with 2048 features
[INFO] - MobileNet loaded with 1280 features
[INFO] Models cache saved to /app/models_cache/
[INFO] Redis connection established
[INFO] Default users initialized with hashed passwords
[SUCCESS] Application ready on port 5000

[LOGIN]     Login successful: admin
[SEARCH] Individual search VGG16 completed in 2.34s
[SEARCH] Combined search (3 models) completed in 6.12s
[CACHE] Redis hit rate: 78%
[PERFORMANCE] Memory usage: 3.2 GB
[SECURITY]        Auto-migrating password for user: researcher
[LOGOUT]        User admin logged out
[HEALTH] All services healthy

[DOCKER] Container multimedia_flask: running
[DOCKER] Container multimedia_redis: running
[DOCKER] Container multimedia_nginx: running
[VOLUMES] redis_data: 45MB, models_cache: 2.1GB, nginx_cache: 128MB
```

Listing 5: Logs de fonctionnement du système

9 Conclusion

Ce projet représente une réalisation technique qui répond aux exigences académiques tout en intégrant des bonnes pratiques de développement moderne. L'intégration de multiples modèles d'IA avec des méthodes de fusion sophistiquées crée un système fonctionnel et robuste. **Points forts de l'implémentation :**

- Architecture microservices bien structurée
- Sécurité de base avec hachage cryptographique des mots de passe
- Interface utilisateur professionnelle et intuitive
- Persistance des données avec volumes Docker
- Déploiement automatisé et reproductible

- Performance satisfaisante pour un environnement académique

Apprentissages et perspectives :

Cette réalisation a permis de comprendre les défis du développement d'applications multimédia complètes, depuis l'implémentation des algorithmes d'IA jusqu'au déploiement cloud sécurisé. Les résultats obtenus valident l'approche multi-modèles et ouvrent des perspectives pour de futurs développements.

Cette expérience démontre qu'un projet académique peut intégrer des standards professionnels tout en constituant un excellent support d'apprentissage pour les technologies modernes d'IA et de cloud computing.

Remerciements

Nous tenons à remercier l'équipe pédagogique du cours "Cloud and Edge computing" pour leur encadrement, ainsi que l'Université de Mons pour avoir fourni l'infrastructure nécessaire à la réalisation de ce projet.