



الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي والبحث العلمي
Ministère de l'Enseignement Supérieure et de la Recherche Scientifique

جامعة وهران 2 محمد بن أحمد
Université d'Oran 2 Mohamed Ben Ahmed

معهد الصيانة و الأمن الصناعي
Institut de Maintenance et de Sécurité Industrielle

Département de Maintenance de l'instrumentation et l'automatisation

MÉMOIRE

Pour l'obtention du diplôme de Master

Filière : Génie industrielle

Spécialité : Maintenance de l'instrumentation et l'automatisation

Thème

Systeme de détection d'incendie visuel

Présenté et soutenu publiquement par :

BOURAS Abderrahmane & BENDJABER Abdeldjalil

Devant le jury composé de :

Nom et Prénom	Grade	Etablissement	Qualité
Mme HAIMOUR Rachida	MCB	IMSI-UNIV2	Présidente
Mme BENABED Khadidja	MCB	IMSI-UNIV2	Examinatrice
Mme DJELGHOUM Farida	MAA	IMSI-UNIV2	Encadreur
Mr BELABBES Abdallah	MCA	IMSI-UNIV2	Co-Encadreur

Année 2025/2026

REMERCIEMENTS

« La gratitude est la mémoire du cœur »

Nous remercions en premier lieu Allah, le Tout-Puissant, de nous avoir accordé la santé, la volonté et la persévérance nécessaires pour mener à bien ce travail.

Nous tenons à exprimer notre profonde gratitude à nos directeurs de mémoire, pour leur encadrement bienveillant, leurs précieux conseils et leur disponibilité tout au long de la réalisation de ce projet. Leur rigueur scientifique et leurs orientations éclairées ont été déterminantes dans l'aboutissement de ce travail.

Nos sincères remerciements vont également aux membres du jury qui ont accepté d'évaluer ce mémoire et de nous faire bénéficier de leurs remarques constructives.

Nous remercions l'ensemble des enseignants de l'Institut de Maintenance et de Sécurité Industrielle d'Oran Université Oran 02, Mohamed Ben Ahmed qui ont contribué à notre formation tout au long de notre cursus universitaire.

Enfin, nos plus chaleureux remerciements s'adressent à nos familles pour leur soutien indéfectible, leur patience et leurs encouragements constants. Ce travail leur est dédié.

RESUME

Ce mémoire présente la conception et la réalisation d'un système de détection d'incendie par intelligence artificielle, baptisé FireDetect Pro, développé dans le cadre d'un projet de fin d'études en Génie Industriel à l'Université Oran 02.

Face aux limitations structurelles des systèmes de détection d'incendie conventionnels (fausses alarmes fréquentes, détection tardive, absence de localisation spatiale précise) une approche innovante fondée sur la vision par ordinateur et l'apprentissage profond a été adoptée. Le système repose sur le modèle YOLOv8, entraîné par transfer learning sur un dataset diversifié d'environ 11000 images annotées représentant des scènes d'incendie variées.

L'architecture logicielle repose sur une structure modulaire multi-threadée permettant la surveillance simultanée de plusieurs caméras en temps réel. Les principaux composants intègrent OpenCV pour le traitement vidéo, PyQt6 pour l'interface graphique, FastAPI pour l'exposition d'une API REST, et un module de notification automatique par e-mail. Une adaptation dynamique jour/nuit et une localisation spatiale par grille 3×3 complètent les fonctionnalités du système.

Les résultats obtenus sont satisfaisants : une précision de 92 %, un rappel de 88 % et un mAP@0.5 de 91 %, avec une vitesse d'inférence de 35 ms par image sur processeur standard sans GPU dédié. Le système a été validé sur des scènes réelles variées (feux intérieurs, extérieurs, nocturnes) avec une stabilité confirmée sur 2 heures de fonctionnement continu.

Ce travail démontre que l'intelligence artificielle appliquée à la vision par ordinateur constitue une solution complémentaire crédible aux systèmes traditionnels, offrant une détection précoce des flammes, une localisation spatiale précise et une intégration aisée dans les infrastructures existantes.

ABSTRACT

This thesis presents the design and implementation of an artificial intelligence-based fire detection system, named FireDetect Pro, developed as part of a final-year project in Industrial Engineering at the University of Oran 02.

Conventional fire detection systems suffer from well-known structural limitations, including frequent false alarms, delayed detection, and lack of precise spatial localization. To address these shortcomings, an innovative approach based on computer vision and deep learning was adopted. The system relies on the YOLOv8 model, trained via transfer learning on a diverse dataset of approximately 11,000 annotated images representing various fire scenarios.

The software architecture is built on a multi-threaded modular structure enabling simultaneous real-time monitoring of multiple camera feeds. Key components include OpenCV for video processing, PyQt6 for the graphical user interface, FastAPI for REST API exposure, and an automatic email notification module. Dynamic day/night adaptation and spatial fire localization using a 3×3 grid further enhance the system's capabilities.

The results achieved are compelling: 92% precision, 88% recall, and a mAP@0.5 of 91%, with an inference speed of 35 ms per frame on a standard CPU without a dedicated GPU. The system was validated on a variety of real-world scenes indoor, outdoor, and nighttime fires with confirmed stability over 2 hours of continuous operation.

This work demonstrates that AI-driven computer vision represents a credible and complementary solution to traditional fire detection systems, offering early flame detection, precise spatial localization, and seamless integration with existing infrastructures.

ملخص

تُقدّم هذه المذكرة تصميم وإنجاز نظام للكشف عن الحريق باستخدام الذكاء الاصطناعي، أطلق عليه اسم FireDetect Pro، وقد طُوّر في إطار مشروع التخرج لنيل شهادة الماستر في الهندسة الصناعية، بجامعة وهران 02.

في مواجهة القيود الهيكلية لأنظمة الكشف عن الحريق التقليدية — كثرة الإنذارات الكاذبة، والكشف المتأخر، وغياب التحديد المكاني الدقيق — تم اعتماد نهج مبتكر يعتمد على رؤية الحاسوب والتعلم العميق. يركز النظام على نموذج YOLOv8، الذي تم تدريبه عبر تقنية نقل التعلم على مجموعة بيانات متنوعة تضم ما يقارب 11,000 صورة موسومة تمثل سيناريوهات حرائق مختلفة.

يقوم النظام على بنية برمجية معيارية متعددة الخيوط تتيح مراقبة عدة كاميرات في وقت واحد وفي الزمن الحقيقي. تشمل المكونات الرئيسية: OpenCV لمعالجة الفيديو، وPyQt6 لواجهة المستخدم الرسومية، وFastAPI لعرض واجهة برمجية REST، فضلاً عن وحدة إرسال إشعارات تلقائية عبر البريد الإلكتروني. كما يتضمن النظام تكيفاً ديناميكياً بين وضعي النهار والليل، وتحديداً مكانياً للحريق عبر شبكة 3×3.

أظهرت النتائج المحققة أداءً متميزاً: دقة بلغت 92%، واستدعاء بلغ 88%، ومعدل $mAP@0.5$ بلغ 91%، مع سرعة استدلال تبلغ 35 ميلي ثانية للإطار الواحد على معالج اعتيادي دون الحاجة إلى وحدة معالجة رسومية مخصصة. وقد تم التحقق من النظام على مشاهد حقيقية متنوعة (حرائق داخلية، وخارجية، وليلية)، مع تأكيد استقراره عبر ساعتين من التشغيل المستمر.

يُثبت هذا العمل أن الذكاء الاصطناعي المُطبّق على رؤية الحاسوب يُمثّل حلاً تكملياً موثقاً لأنظمة الكشف التقليدية، إذ يوفر كشفاً مبكراً للنيران، وتحديداً مكانياً دقيقاً، وتكاملاً سلساً مع البنى التحتية الق

SOMMAIRE

REMERCIEMENTS.....	II
SOMMAIRE	vi
LISTE DES FIGURES.....	xi
INTRODUCTION GENERALE	1
CHAPITRE I :	3
Étude scientifique sur le feu et état de l'art.....	3
Introduction.....	4
I.1 Étude scientifique du feu	5
I.1.1 Définition physique du feu	5
I.1.2 Mécanisme de propagation	6
I.1.3 Phases de développement d'un incendie.....	7
I.2 Classes de feu	8
I.3 Description du Système de Détection d'Incendie	9
I.3.1 Définition	9
I.3.2 Composants Généraux d'un SDI	10
I.3.3 Principe de Fonctionnement Global	11
I.4 Les Systèmes de Détection Traditionnels	11
I.4.1 Détecteurs de Fumée	12
I.4.2 Détecteurs de Chaleur (Thermostatiques)	14
I.4.3 Détecteurs de Flamme (Optiques Infrarouges/UV)	14
I.4.4 Détecteurs de Gaz Combustibles	15
I.5 Limites et Inconvénients des Systèmes Traditionnels.....	16
I.5.1 Sensibilité aux fausses alarmes	16
I.5.2 Limites dans les environnements complexes	17
I.5.3 Absence de localisation précise	18
I.5.4 Détection Tardive par Nature.....	18
I.6 De la Détection Conventionnelle vers la Vision par Intelligence Artificielle	18
I.6.1 Contexte et Motivation.....	18
I.6.2 Solutions Existantes dans le Domaine	19
I.6.3 Apport de l'Apprentissage Profond (Deep Learning).....	20
I.6.4 Comparaison entre systèmes Traditionnels et détection Visuelle intelligente	21
Conclusion	22
CHAPITRE II :	23
Outils et technologies utilisées	23

Introduction.....	24
II.1 Python : Langage de Développement	24
II.1.1 Présentation et Historique de Python	24
II.1.2 Python et l'Intelligence Artificielle.....	27
II.2 YOLO	27
II.2.1 Contexte et Motivation	27
II.2.2 Principe Fondamental	28
Étape 3 : Calcul du score final	29
II.2.3 Évolution de la Famille YOLO.....	30
II.3 Traitement d'Image et Vision par Ordinateur (OpenCV).....	31
II.3.1 Présentation	32
II.3.2 Rôle dans ce Projet	32
II.4 Framework d'Interface Graphique (PyQt6)	33
II.4.1 Mécanisme Central.....	33
II.5 Les Caméras Utilisées dans les Systèmes de Détection d'Incendie	33
II.5.1 Déploiement des Caméras de Surveillance en Algérie	37
II.5.2 Recommandation de camera pour le déploiement Optimal du Système	38
II.5 Récapitulatif des Technologies.....	39
Conclusion	39
CHAPITRE III :	40
Conception et Réalisation et Application.	40
Introduction.....	41
III.1.3 Fonctionnement général en temps réel	44
III.2 Acquisition du Dataset d'Entraînement.....	46
III.2.1 Collecte et Composition des Données	46
III.2.2 Annotation des Données au Format YOLO	47
III.2.3 Augmentation des Données.....	48
III.2.4 Répartition des Dataset	48
III.3 Entraînement du Modèle YOLOv8.....	48
III.3.1 Apprentissage par transfert.....	48
III.3.2 paramètres d'Entraînement.....	49
III.3.3 Résultats obtenus.....	49
III.4 Adaptation Dynamique Jour/Nuit.....	50
III.5 Développement des Modules Fonctionnels.....	51
III.5.1 Gestion des Caméras.....	52
III.5.2 Module de détection et moteur d'Inférence.....	52
III.5.3 Localisation par Grille 3×3	54
III.5.4 Carte thermique des détections	55

III.5.5 Système d'Alertes	56
III.5.6 Enregistrement Vidéo	57
III.6 Interface graphique	58
III.6.1 Architecture de l'Interface.....	58
III.6.2 Sidebar : État Global du Système.....	59
III.6.3 Onglet Heatmap	59
III.6.4 Onglet de configuration avancée	59
III.7 API REST.....	60
III.7.1 Conception.....	60
III.8 Optimisations de Performance	61
III.9 Installation et Déploiement.....	62
III.9.1 Les prérequis du Système	62
III.9.2 Procédure d'Installation	63
III.9.3 Compatibilité Multi-Plateformes.....	64
III.10 Vue d'Ensemble de l'architecture du système de notification	64
III.11 Suivi de la Durée du Feu.....	66
III.11.3 Logique par Image	67
III.12 Annotation de L'image	67
III.12.1 Objectif	67
III.12.2 Pipeline d'Annotation en Trois parties.....	67
III.13 Module de notification par e-mail	68
III.13.1 Conception de la Classe.....	68
III.13.2 Mécanisme de Sécurité des Fils d'Exécution	69
III.13.3 Validation Avant Envoi.....	69
III.13.5 Envoi de l'email avec pièce jointe	70
III.14 Configuration et Intégration des Paramètres	70
III.14.1 Paramètres Configurables par l'Utilisateur	71
III.14.2 Garantie des Identifiants au Démarrage	72
III.16 Limites et Travaux Futurs.....	73
III.17 Tests et Validation du Système.....	73
III.17.1 Tests Unitaires des Modules.....	74
III.17.2 Tests en environnement local.....	74
III.17.3 Résultats des Tests.....	74
Conclusion	76
III.18 Perspectives d'évolution	77
CONCLUSION GENERALE.....	78
BIBLIOGRAPHIE.....	79
ANNEXE	81

Annexe A : Spécifications Techniques du Système FireDetect Pro	81
Annexe B : Guide d'Installation et de Configuration	83
Annexe C : Résultats Détaillés des Tests de Performance	84

LISTE DES FIGURES

Figure I.1 triangle de feu	5
FigureI.2 Mode de propagation du feu.....	6
Figure I.3 les classes de feu	9
Figure I.4 les composants généraux d'un SDI	11
Figure I.5 détecteur ionique.....	12
Figure I.6 détecteur optique.....	13
Figure I.7 détecteur de chaleur	14
Figure I.8 les détecteurs de flammes.....	15
Figure I.9 détecteur de gaz combustible.....	16
Figure II.1 Popularité des langages de programmation	26
Figure II.2 Approche de détection	28
FigureII.3 l'architecture de YOLOv8	31
Figure II.4Traitement d'image et vision par ordinateur	32
FigureII.5Camera IP.....	35
FigureII.6Camera thermique	36
Figure II.7Camera IP avec IA embarquée.....	37
Figure III.1pipeline multi-thrèade du système flux de données entre les 5 threads	45
Figure III.2GRILE 3x3.....	55
Figure III.3Interface Graphique.....	59
Figure III.4Onglets parametres.....	60
Figure III.5 : Algorithme de déclenchement de l'alerte e-mail basé sur la durée de détection	67
FigureIII.6 Fonction <code>_build_email_snapshot()</code> construction de l'image annotée en trois couches graphiques	68
Figure III.7 : Mécanisme de verrouillage threading.Lock pour la prévention des envois simultanés	69
FigureIII.8 : Implémentation de la livraison SMTP sécurisée avec chiffrement STARTTLS.....	70
Figure III.9Parametres configurables pour alertes email	72
FigureIII.10 Garantie des paramètres SMTP à l'initialisation du système.....	72
FigureIII.11 Séquence d'envoi de l'alerte e-mail de la détection à la confirmation SMTP.....	73
Figure III.12l'envoi automatique du mail.....	75
Figure III.13détection du feu	75
Figure III.14App mobile FireDetect Pro	77

Liste des Tableaux

Tableau I.1 comparaison entre détecteur ionique et détecteur optique	13
Tableau I .2 limitations physiques dans les environnements complexes	17
Tableau I.3 Comparaison entre systèmes traditionnels et détection visuelle intelligente	21
Tableau II.1Caracteristiques et leurs descriptions	26
Tableau II.2évolution de la famille YOLO	30
Tableau II.3 fonction de chaque composant du projet	32
Tableau II.4Comparaison entre les types de camera	34
Tableau II.5 options de caméras recommandées par niveau de budget	38
Tableau II.6 Caractéristique des outils	39
Tableau II.1 Architecture des modules	42
TableauIII.2composition et sources du Dataset d'entrainement	46
Tableau III.3 les paramètres d'entrainement et leur justifications	49
Tableau III.4Métriques de performance évaluées sur l'ensemble de test.....	50
Tableau III.5Endpoints de l'API REST et exemples de réponses JSON	61
Tableau III.6optimisations de performance et leur impact mesuré	62
Tableau III.7variables d'instance	66
Tableau III.8pipeline d'annotation en trois couches	68
Tableau III.9 conception de la classe EMAILNOTIFER.....	68
Tableau III.10classification de la sévérité	69
Tableau III.11 paramètres configurables par l'utilisateur	71
Tableau III.12 limites et travaux futurs.....	73

LISTE DES ABRÉVIATIONS

Sigle	Signification
IA	Intelligence Artificielle
CNN	Convolutional Neural Network (Réseau de Neurones Convolutif)
YOLO	You Only Look Once
YOLOv8	You Only Look Once — version 8
SDI	Système de Détection d'Incendie
CMSI	Centralisateur de Mise en Sécurité Incendie
ECS	Équipement de Contrôle et de Signalisation
API	Application Programming Interface (Interface de Programmation Applicative)
REST	Representational State Transfer
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
CPU	Central Processing Unit (Unité Centrale de Traitement)
GPU	Graphics Processing Unit (Processeur Graphique)
FPS	Frames Per Second (Images par Seconde)
RGB	Red, Green Blue (Rouge Vert Bleu)
IR	Infrarouge
UV	Ultraviolet
mAP	Mean Average Precision (Précision Moyenne)
RTSP	Real Time Streaming Protocol
BMS	Building Management System (Système de Gestion du Bâtiment)
SCADA	Supervisory Control and Data Acquisition
IoT	Internet of Things (Internet des Objets)
PWA	Progressive Web App (Application Web Progressive)
COCO	Common Objects in Context (dataset de référence)
FEMA	Federal Emergency Management Agency
RGPD	Règlement Général sur la Protection des Données
ERP	Établissement Recevant du Public

ICPE	Installation Classée pour la Protection de l'Environnement
CNPP	Centre National de Prévention et de Protection
ONNX	Open Neural Network Exchange
OpenCV	Open-Source Computer Vision Library
PyQt6	Python Bindings for Qt6 Framework
FastAPI	Fast Application Programming Interface (framework Python)
NumPy	Numerical Python
PIB	Produit Intérieur Brut

INTRODUCTION GENERALE

La sécurité incendie constitue l'un des enjeux les plus critiques des sociétés modernes. Phénomène destructeur par nature, l'incendie se développe avec une rapidité redoutable: en l'absence d'une détection précoce, une simple flamme peut évoluer en sinistre incontrôlable en l'espace de quelques minutes seulement. Les conséquences humaines, économiques et environnementales qui en résultent sont considérables. À l'échelle mondiale, les pertes directes liées aux incendies représentent environ 1 % du PIB mondial chaque année, tandis que les statistiques de la FEMA recensent, pour la seule année 2023 aux États-Unis, près de 1 389 000 incendies ayant causé 3 670 décès et 23,2 milliards de dollars de pertes directes.

Face à ce constat alarmant, les systèmes de détection d'incendie (SDI) conventionnels reposant sur des capteurs de fumée, de chaleur ou de flamme constituent depuis des décennies le principal rempart contre ce risque. Cependant, ces technologies présentent des limitations structurelles bien connues : fausses alarmes fréquentes, détection tardive dans les grands espaces ou les environnements difficiles, et surtout une incapacité totale à localiser précisément la zone du sinistre. Ces failles peuvent s'avérer fatales dans des contextes industriels ou à forte densité humaine.

Parallèlement, l'essor de l'intelligence artificielle et, plus particulièrement, des techniques de vision par ordinateur et d'apprentissage profond, ouvre de nouvelles perspectives dans le domaine de la détection automatique. Des modèles comme YOLOv8 permettent désormais d'analyser en temps réel des flux vidéo et d'identifier visuellement la présence de flammes avec une précision et une rapidité que les capteurs physiques traditionnels ne peuvent pas atteindre.

C'est dans ce contexte que s'inscrit le présent travail, réalisé dans le cadre d'un projet de fin d'études. Il soulève la problématique centrale suivante :

Dans quelle mesure est-il possible de concevoir un système de détection d'incendie basé sur l'intelligence artificielle et la vision par ordinateur, capable de pallier les limites des systèmes traditionnels en offrant une détection précoce, une localisation spatiale précise des flammes et une surveillance multi-caméra en temps réel ?

Pour répondre à cette question, ce mémoire propose la conception et la réalisation de FireDetect Pro, un système modulaire s'appuyant sur le modèle YOLOv8, entraîné par transfer learning sur un dataset de près de 11 000 images annotées, et intégrant une architecture logicielle multi-threadée, une interface graphique PyQt6, une API REST via FastAPI, ainsi qu'un module de notification automatique.

Le présent document est organisé en trois chapitres. Le premier chapitre établit une étude scientifique du phénomène du feu, passe en revue les systèmes de détection traditionnels et leurs limites, et présente l'apport de l'intelligence artificielle dans ce domaine. Le deuxième chapitre décrit les outils et technologies retenus pour le développement du système. Enfin, le troisième chapitre détaille la conception, le développement et la validation du système FireDetect Pro.

CHAPITRE I :

Étude scientifique sur le feu et état de l'art.

Introduction

La sécurité incendie représente l'un des défis les plus critiques auxquels sont confrontées les sociétés modernes, aussi bien dans les milieux industriels, résidentiels qu'environnementaux. L'incendie est un phénomène destructeur qui se développe avec une rapidité redoutable en l'absence d'une détection précoce, un feu peut passer du stade de la simple flamme à un sinistre incontrôlable en quelques minutes seulement. Les conséquences humaines, économiques et environnementales qui en résultent sont considérables.

À l'échelle mondiale, les pertes directes liées aux incendies sont estimées à environ 1 % du PIB mondial chaque année [1]. Les incendies de bâtiments, combinés aux feux de forêt et d'espaces industriels, sont responsables d'environ 180 000 décès annuels à travers le monde selon le Global Fire Statistics Centre. Selon un rapport de la Banque mondiale publié en 2025, la pollution atmosphérique générée par les seuls feux de forêt serait en outre responsable de 1,5 million de décès supplémentaires par an, un chiffre alarmant qui illustre l'ampleur du problème au-delà des destructions directes [2]. Aux États-Unis, les données de la FEMA recensent pour l'année 2023 près de 1 389 000 incendies, ayant causé 3 670 décès et généré 23,2 milliards de dollars de pertes directes [3].

En Algérie, la situation est tout aussi préoccupante. Selon les données relayées par Atlas Magazine en octobre 2025, 811 incendies se sont déclarés sur l'ensemble du territoire national au cours de la seule saison estivale 2025, touchant 32 wilayas et ravageant des milliers d'hectares de forêts et de zones habitées[4]. Ces statistiques dramatiques confirment l'urgence absolue de disposer de systèmes de détection plus réactifs et plus fiables.

Face à ce constat, les systèmes de détection d'incendie jouent un rôle déterminant : chaque seconde gagnée entre l'apparition du foyer et l'alerte peut faire la différence entre un début d'incendie maîtrisable et une catastrophe.

Nous allons démontrer dans ce chapitre, les systèmes conventionnels qui sont encore largement déployés, présentent des limitations structurelles importantes : fausses alarmes, couverture restreinte, détection tardive qui compromettent leur efficacité dans de nombreux contextes réels. Ce chapitre a pour objectif de présenter les fondements théoriques des systèmes de détection d'incendie existants, d'analyser leurs forces et leurs faiblesses, puis de

justifier le choix d'une approche par vision par ordinateur et intelligence artificielle comme solution complémentaire et innovante pour une détection plus précoce, plus précise et plus adaptable.

I.1 Étude scientifique du feu

I.1.1 Définition physique du feu

Le feu est une réaction d'oxydation exothermique rapide libérant :

- La chaleur (énergie thermique)
- Les gaz de combustion
- La lumière (flammes)

Fire Triangle

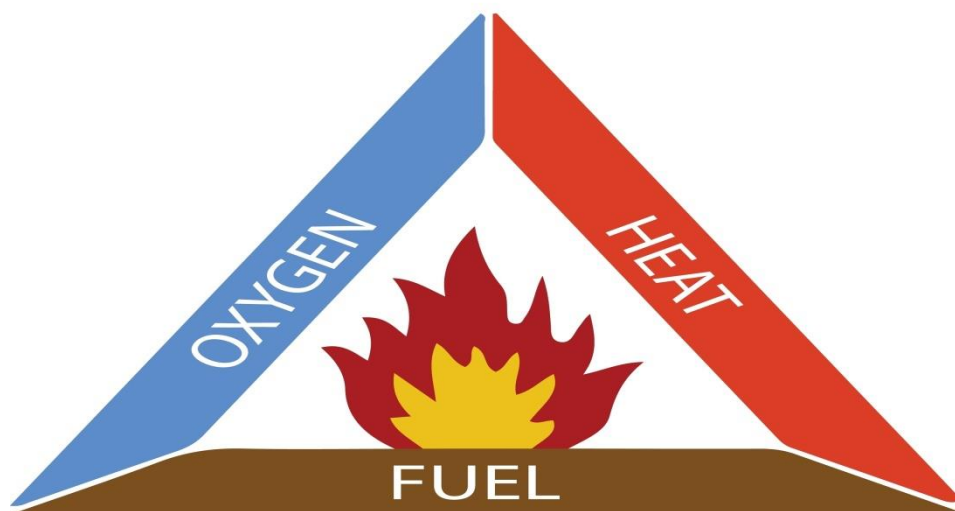


Figure I.1 triangle de feu

Il est régi par le triangle du feu présenter par la figure 1 qui illustre les éléments suivants :

- Combustible (bois, gaz, liquide, métal)
- Comburant (oxygène)

- Énergie d'activation

La suppression d'un seul élément entraîne l'extinction du feu.

I.1.2 Mécanisme de propagation

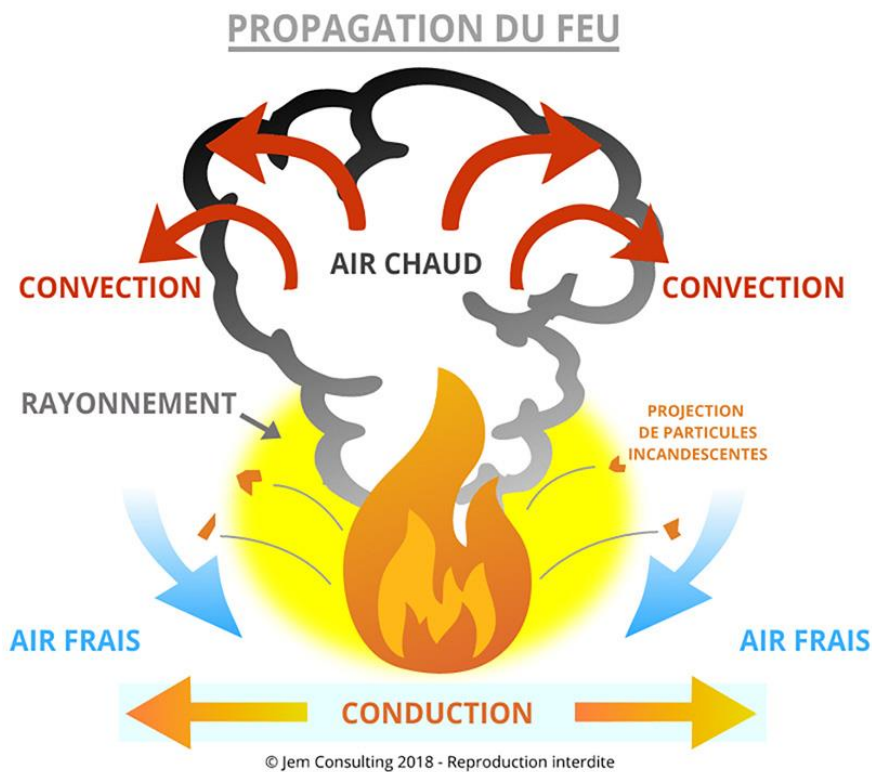


Figure I.2 Mode de propagation du feu

La propagation d'un incendie est régie par quatre modes de transfert thermique qui agissent souvent de manière simultanée :

- **La conduction** : Transfert de chaleur à travers les solides par contact direct. Une structure métallique chauffée peut ainsi enflammer des matériaux éloignés de la source initiale.
- **La convection** : Déplacement des gaz chauds et des fumées vers le haut. Dans les bâtiments, ce phénomène favorise l'accumulation de chaleur sous les plafonds et la

diffusion rapide des fumées, ce qui augmente la température ambiante et peut entraîner l'inflammation de matériaux situés à distance du foyer.

- **Le rayonnement** : Propagation de la chaleur par ondes infrarouges, sans contact direct. Un foyer intense peut ainsi échauffer puis enflammer des matériaux situés à plusieurs mètres de distance, ce qui représente un danger important, notamment dans les espaces industriels ou ouverts.
- **La projection** : Transport de particules, braises, gouttelettes ou matières enflammées vers d'autres zones. Ce mode de propagation peut provoquer l'apparition de nouveaux foyers à distance, en particulier sous l'effet du vent, des courants d'air ou de l'éclatement de certains matériaux.

Plusieurs facteurs aggravent cette propagation comme la nature et la sécheresse du combustible, l'apport en oxygène par ventilation ou courants d'air, la charge calorifique de l'espace, et la configuration des lieux (couloirs, gaines, faux plafonds) qui constituent des voies de propagation privilégiées.

I.1.3 Phases de développement d'un incendie

Un incendie évolue généralement en quatre phases :

1. Phase d'ignition

C'est la phase de départ qui est le moment précis où l'énergie d'activation, qu'il s'agisse d'une étincelle, d'une flamme ou d'une forte chaleur, rencontre le combustible en présence d'oxygène. À ce stade, le feu est encore très localisé et la température commence à peine à monter. C'est précisément durant cette phase qu'il est le plus facile d'intervenir et d'éteindre le feu avec un extincteur.

2. Phase de croissance

C'est la phase de développement où le feu commence à se propager aux objets environnants. La température augmente rapidement tandis que des fumées chaudes s'accumulent au plafond, et le feu nécessite de plus en plus d'oxygène pour se maintenir. Un signe distinctif de cette phase est que la chaleur rayonnée par les flammes commence à chauffer les autres meubles et matériaux à distance, sans contact direct.

3. Phase de flashover

C'est l'étape d'embrasement généralisé. Ce n'est pas une phase longue, mais un moment de transition brutale. La chaleur accumulée dans la pièce devient alors si forte, autour de 600°C, que tous les objets inflammables s'enflamment simultanément sans même être touchés par les flammes. À ce stade, la pièce devient totalement invivable en quelques secondes.

4. Phase de décroissance

C'est l'étape d'extinction du feu, une fois que le pic de chaleur est passé. Le feu baisse d'intensité soit parce qu'il n'y a plus de combustible, lorsque tout a brûlé, soit parce qu'il n'y a plus assez d'oxygène en raison du confinement. Même si les flammes diminuent, le risque reste important, car les braises demeurent très chaudes et peuvent provoquer une reprise de feu si de l'air frais entre brusquement.

I.2 Classes de feu

Les incendies sont classifiés selon la nature du combustible présenté par la figure I.3

CLASSES DE FEU SELON LES NORMES INTERNATIONALES					
CLASSE DE FEU	SYMBOLE	DESCRIPTION	EXEMPLES	CARACTÉRISTIQUES	MOYENS D'EXTINCTION RECOMMANDÉS
A		Solides (incluant les matériaux organiques)	Bois, papier, carton, tissu, charbon	Se consomment en formant des braises	Eau, mousse, poudre ABC
B		Liquides inflammables	Essence, alcool, huiles, peintures, solvants	Brûlent à la surface et ne laissent pas de braises	Mousse, poudre BC, CO ₂
C		Gaz inflammables	Propane, butane, méthane, gaz naturel	Se propagent rapidement et peuvent exploser	Poudre BC, CO ₂
D		Métaux combustibles	Aluminium, magnésium, titane, sodium	Réagissent violemment avec l'eau et l'air	Poudre spéciale D
F		Huiles et graisses de cuisson	Huiles végétales ou animales (huile de friture)	Brûlent à haute température et peuvent provoquer des projections	Agent chimique humide, extincteur spécial F

Figure I.3 les classes de feu

Les classifications des feux selon la nature du combustible se résument comme suite :

Classe A : Matières solides (bois, papier, textiles)

Classe B : Liquides inflammables (essence, solvants, huiles)

Classe C : Gaz inflammables (butane, propane, méthane)

Classe D : Métaux combustibles (magnésium, sodium, aluminium)

Classe F : Les huiles et graisses de cuisson

I.3 Description du Système de Détection d'Incendie

I.3.1 Définition

Un système de détection d'incendie (SDI) est un ensemble de dispositifs électroniques interconnectés, dont la fonction principale est d'identifier les signes avant-coureurs d'un incendie élévation de température anormale, présence de fumée, apparition de flammes visibles, ou émanation de gaz combustibles et de déclencher automatiquement une alarme ou une action de sécurité appropriée. Son objectif fondamental se résume en trois axes :

- **La protection des personnes** : permettre une évacuation rapide et ordonnée avant que le sinistre ne se propage

- **La protection des biens** : limiter les dommages matériels par une intervention précoce
- **La protection de l'environnement** : prévenir la diffusion de fumées toxiques et de polluants [5].

I.3.2 Composants Généraux d'un SDI

Un système de détection d'incendie classique est composé de quatre éléments fondamentaux :

1. Les détecteurs : C'est des capteurs placés dans les zones à surveiller, ils mesurent en continu les paramètres physiques ou chimiques de l'environnement (température, densité optique de fumée, ionisation, concentration de gaz). Ils constituent le premier maillon de la chaîne de détection.

2. Le tableau de signalisation : C'est une centrale d'incendie qui est le cerveau du système, il reçoit les signaux envoyés par les détecteurs, les analyse, et détermine si un état d'alarme doit être déclenché. Il gère également la signalisation des défauts, la mise en sécurité et la traçabilité des événements.

3. Les dispositifs d'alarme : Une fois l'état d'alarme confirmé par la centrale, celle-ci active les dispositifs d'alerte (sirènes sonores, flashes lumineux, notifications vocales, ou transmissions vers un poste de surveillance distant).

4. Les actionneurs de sécurité : Dans les installations industrielles ou les grands bâtiments, le SDI peut commander des équipements actifs tels que les sprinklers, les clapets de désenfumage, les portes coupe-feu ou les extincteurs automatiques

Tous ces composants sont illustrés sur la figure I.4

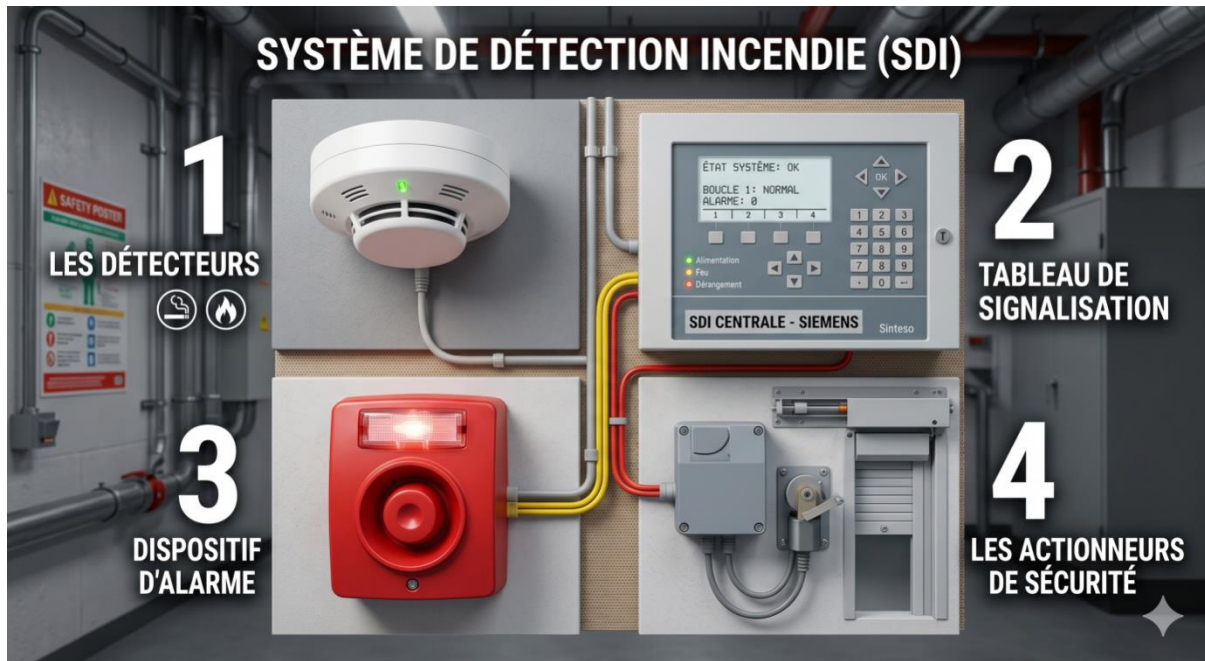


Figure I.4 les composants généraux d'un SDI

I.3.3 Principe de Fonctionnement Global

Le fonctionnement d'un SDI repose sur un cycle de surveillance permanent. Les capteurs analysent continuellement les paramètres de leur environnement. Dès qu'une valeur mesurée dépasse un seuil prédéfini, un signal d'alarme est transmis à la centrale. Celle-ci traite ce signal parfois en le croisant avec d'autres capteurs pour éviter les fausses alarmes puis active les dispositifs d'alerte et de mise en sécurité.

Le paramètre critique qui détermine l'efficacité de tout le système est le temps de détection : c'est l'intervalle qui s'écoule entre l'instant où le feu commence à se développer et le moment où l'alarme se déclenche. Plus ce délai est court, plus les chances de maîtriser le sinistre sans conséquences graves sont élevées.

I.4 Les Systèmes de Détection Traditionnels

Les systèmes de détection d'incendie traditionnels peuvent être classifiés selon le phénomène physique ou chimique qu'ils détectent.

I.4.1 Détecteurs de Fumée

Les détecteurs de fumée sont les dispositifs les plus répandus dans les installations résidentielles et tertiaires. Deux technologies principales coexistent :

a) Détecteurs ioniques

Basés sur une chambre d'ionisation contenant une source radioactive faiblement active, généralement de l'Américium-241, ils maintiennent un courant d'ionisation constant en conditions normales. Lorsque des particules de combustion pénètrent dans la chambre, elles perturbent ce courant et déclenchent l'alarme. Ils sont particulièrement sensibles aux feux vifs à combustion rapide. Ce type de détecteur est surtout utilisé en intérieur, dans les logements, bureaux et locaux fermés.



Figure I.5 détecteur ionique

b) Détecteurs optiques (photoélectriques)

Ils utilisent un émetteur infrarouge et un récepteur placé dans une chambre de mesure. En l'absence de fumée, le faisceau lumineux n'atteint pas le récepteur. Lorsque des particules de fumée entrent dans la chambre, elles diffusent la lumière vers le récepteur, provoquant l'alarme. Ces détecteurs sont mieux adaptés aux feux couvants à combustion lente. Ils sont également destinés principalement à un usage intérieur, notamment dans les bâtiments résidentiels, commerciaux et administratifs.



Figure I.6 détecteur optique

Et le tableau I.1 va nous montrer la différence entre les deux détecteurs

Tableau I.1 comparaison entre détecteur ionique et détecteur optique

Caractéristique	Détecteur Ionique	Détecteur Optique
Sensibilité optimale	Feux vifs, flammes nues	Feux couvants, fumée dense
Sensibilité aux fausses alarmes	Élevée (vapeur, aérosols)	Moyenne (poussière, insectes)
Portée de détection	Limitée (proximité du capteur)	Limitée (proximité du capteur)
Localisation précise du foyer	Non	Non
Coût unitaire	Très faible	Faible à moyen

I.4.2 Détecteurs de Chaleur (Thermostatiques)

Les détecteurs thermiques mesurent directement la température ambiante et se présentent en deux variantes :

- **Détecteurs à seuil fixe** : se déclenchent lorsque la température atteint une valeur prédéfinie, généralement comprise entre 57°C et 90°C selon les normes d'installation.
- **Détecteurs thermos différentiels** : réagissent à une vitesse d'augmentation de température anormale, typiquement supérieure à 10°C par minute, indépendamment de la température absolue.



Figure I.7 détecteur de chaleur

Ces détecteurs présentent moins de sensibilité aux fausses alarmes que les détecteurs de fumée, mais leur temps de réponse est intrinsèquement plus long le feu doit déjà être suffisamment établi pour provoquer une élévation de température mesurable au niveau du capteur [8], ils sont donc particulièrement adaptés aux environnements intérieurs difficiles, tels que les cuisines, ateliers, garages, chaufferies ou locaux industriels où la fumée peut être présente sans qu'il y ait nécessairement un incendie.

I.4.3 Détecteurs de Flamme (Optiques Infrarouges/UV)

Ces détecteurs analysent le rayonnement lumineux émis spécifiquement par les flammes. Ils utilisent des capteurs sensibles aux longueurs d'onde de l'infrarouge (IR) ou de l'ultraviolet (UV) caractéristiques de la combustion. Leur avantage est une détection très rapide et une

immunité aux faux déclenchements dus à la fumée ou à la chaleur seules. En revanche, leur coût est élevé et leur champ de vision est directionnel ils nécessitent une ligne de vue directe avec la flamme.

Ces détecteurs sont surtout utilisés dans les espaces industriels, les zones à haut risque et parfois en extérieur, notamment dans les raffineries, dépôts de carburant, hangars ouverts ou zones de stockage sensibles.



Figure I.8 les détecteurs de flammes

I.4.4 Détecteurs de Gaz Combustibles

Utilisés principalement dans les environnements industriels (hydrocarbures, méthane, propane, hydrogène), ces capteurs détectent la présence de gaz inflammables avant même l'apparition d'une flamme visible. Ils permettent théoriquement de prévenir l'incendie avant son départ. Toutefois, leur utilisation est limitée à des contextes spécifiques et leur calibrage doit être renouvelé périodiquement.



Figure I.9 détecteur de gaz combustible

I.5 Limites et Inconvénients des Systèmes Traditionnels

Malgré leur large diffusion, les systèmes traditionnels de détection d'incendie présentent plusieurs limites qui réduisent leur efficacité dans de nombreux contextes réels. Leur fonctionnement repose généralement sur la mesure de phénomènes physiques ou chimiques tels que la fumée, la chaleur ou certains gaz, ce qui signifie qu'ils interviennent souvent après que l'incendie a déjà commencé à se développer. Cette caractéristique limite leur capacité à assurer une détection véritablement précoce, notamment dans les environnements complexes.

I.5.1 Sensibilité aux fausses alarmes

L'un des principaux inconvénients de ces systèmes réside dans leur forte sensibilité aux fausses alarmes. La vapeur d'eau, la poussière, les fumées de cuisine, les insectes ou encore certaines variations de température peuvent suffire à déclencher une alerte intempestive. Ces déclenchements non fondés entraînent une perte de confiance des utilisateurs, perturbent les activités normales et mobilisent inutilement les services d'intervention. À long terme, ils peuvent également conduire à une forme de banalisation des alarmes, réduisant ainsi leur efficacité réelle.

I.5.2 Limites dans les environnements complexes

Les établissements industriels sont soumis à des phénomènes environnementaux qui rendent les détecteurs traditionnels particulièrement peu fiables [5] :

Tableau I .2 limitations physiques dans les environnements complexes

Environnement difficile	Phénomène physique	Impact sur la détection
Locaux à grande hauteur (entrepôts, hangars, usines)	Stratification thermique sous toiture	La fumée chaude monte mais se refroidit et se stabilise avant d'atteindre le détecteur en plafond
Zones fortement ventilées (tunnels, ateliers)	Flux aérauliques importants	La fumée est dispersée avant d'atteindre le détecteur
Milieux empoussiérés (traitement des déchets, cimenteries)	Encrassement premature	Fausses alarmes permanentes ou détecteurs obstrués et inactifs
Zones humides ou avec vapeur	Condensation	Déclenchements intempestifs des détecteurs optiques
Environnements très lumineux (verrières)	Saturation optique	Perturbation des détecteurs photoélectriques
Espaces ouverts ou extérieurs	Absence de confinement	Dispersion totale de fumée détection impossible

Les performances des détecteurs traditionnels sont également affectées par les conditions physiques du milieu. Dans les locaux de grande hauteur, la fumée peut se stratifier sous la toiture sans atteindre rapidement les capteurs placés au plafond. Dans les zones fortement ventilées, elle peut être dispersée avant d’être détectée. De même, les milieux poussiéreux, humides ou très lumineux peuvent provoquer des dysfonctionnements, des encrassements

prématurés ou des déclenchements intempestifs. Ces contraintes rendent ces systèmes moins fiables dans les environnements industriels, les entrepôts ou les espaces ouverts.

I.5.3 Absence de localisation précise

Un détecteur ponctuel traditionnel signale qu'une alarme est active dans une *zone*, mais est incapable de préciser dans *quelle partie* de cette zone se situe le foyer, ni à quelle distance, ni avec *quelle intensité* il se développe. Dans un bâtiment industriel, une zone de détection peut couvrir plusieurs centaines de mètres carrés. Cette imprécision contraint les équipes d'intervention à explorer l'ensemble de la zone avant de localiser et d'attaquer le foyer une perte de temps précieuse dans les premières minutes critiques d'un incendie [7].

I.5.4 Détection Tardive par Nature

Les systèmes basés sur la fumée ou la chaleur interviennent après que le feu est déjà établi. Le détecteur thermique ne réagit qu'une fois la température significativement élevée dans son voisinage immédiat ; le détecteur de fumée doit attendre que la fumée se diffuse physiquement jusqu'à lui. Ces délais intrinsèques peuvent se mesurer en plusieurs dizaines de secondes, voire en minutes dans les grands volumes.

La détection par vision artificielle, en revanche, identifie la flamme dès son apparition initiale et visible, avant même que la fumée ne se propage significativement, offrant un avantage décisif en termes de réactivité [9].

I.6 De la Détection Conventionnelle vers la Vision par Intelligence Artificielle

I.6.1 Contexte et Motivation

L'émergence des nouvelles technologies numériques a profondément transformé les approches de surveillance et de sécurité. Dans le domaine de la détection d'incendie, cette évolution ouvre la voie à des solutions plus rapides, plus précises et plus adaptables que les systèmes conventionnels reposant uniquement sur des capteurs physiques.

Les systèmes traditionnels, bien qu'efficaces dans certains contextes, présentent plusieurs limites lorsqu'ils sont confrontés à des environnements complexes, à de grands volumes ou à des conditions de luminosité variables. Leur fonctionnement repose souvent sur la détection indirecte du feu, notamment à travers la fumée, la chaleur ou certains gaz, ce qui peut retarder

l'alerte ou générer des fausses détections. De plus, ces systèmes ne permettent généralement pas de localiser précisément l'origine du sinistre.

Face à ces limites, la vision par ordinateur et l'apprentissage profond constituent une alternative prometteuse. Grâce à l'analyse directe des images ou des flux vidéo, il devient possible de détecter visuellement les flammes dès leur apparition, tout en fournissant une information plus riche sur la scène observée. Des modèles comme YOLOv8 permettent aujourd'hui d'exploiter ces données en temps réel avec un bon niveau de précision et une rapidité compatible avec les exigences opérationnelles.

Cette approche ne vise pas à remplacer totalement les dispositifs traditionnels, mais plutôt à les compléter par une couche de surveillance intelligente. L'objectif est d'améliorer la réactivité du système, de réduire les fausses alarmes et d'offrir une meilleure compréhension spatiale de l'incendie. Dans cette perspective, l'intelligence artificielle s'impose comme une solution pertinente pour renforcer les dispositifs de sécurité incendie existants.

I.6.2 Solutions Existantes dans le Domaine

La maturité croissante de cette technologie se reflète dans l'émergence de solutions industrielles certifiées :

- Bosch AVIOTEC : système de détection incendie par analyse vidéo IA, dédié aux entrepôts et espaces logistiques[10].
- DEF Fire Eye : premier système de détection vidéo incendie certifié "CNPP Certified" (2019), testé selon les foyers types TF1 à TF8 conformes à la norme EN54, déployé notamment pour Airbus sur ses hangars de Toulouse[5].
- Pyronear : solution open source de détection précoce de feux de forêt par caméra et IA, développée en France[11].
- Securitas France : service de détection incendie par vidéo analytique intégrant l'IA pour la surveillance industrielle[12].

Cependant, la grande majorité de ces solutions sont propriétaires, coûteuses et fermées, nécessitant un matériel spécifique et des contrats de maintenance avec le fournisseur. Elles restent inaccessibles aux petites et moyennes structures, aux établissements publics en pays en développement, et aux projets à budget contraint.

I.6.3 Apport de l'Apprentissage Profond (Deep Learning)

L'émergence des algorithmes d'apprentissage profond, et en particulier des réseaux de neurones convolutifs (CNN), a révolutionné la capacité des machines à interpréter le contenu visuel des images. Ces algorithmes apprennent automatiquement à reconnaître des motifs visuels complexes à partir d'exemples annotés, sans nécessiter de règles programmées manuellement.

Appliqués à la détection d'incendie, ces modèles apprennent à distinguer les flammes de toutes les autres sources lumineuses (soleil, lampes, reflets) en analysant non seulement la couleur, mais aussi la texture, la forme et le mouvement temporel caractéristiques de la combustion. Les architectures modernes comme YOLO (*You Only Look Once*) permettent de réaliser cette analyse en temps réel plusieurs dizaines d'images par seconde sur du matériel standard

I.6.4 Comparaison entre systèmes Traditionnels et détection Visuelle intelligente

Tableau I.3 Comparaison entre systèmes traditionnels et détection visuelle intelligente

Critère	Systèmes Traditionnels	Détection Visuelle par IA
Principe	Capteurs physiques (fumée, chaleur, gaz)	Analyse d'image par réseau de neurones
Moment de détection	Après établissement du feu	Dès apparition de la flamme visible
Localisation du foyer	Zone approximative	Pixel précis dans l'image
Couverture	Un capteur par zone restreinte	Une caméra = plusieurs dizaines de m²
Résistance aux fausses alarmes	Faible (poussière, vapeur, insectes)	Contrôlée par seuil de confiance
Efficacité en grand volume	Très limitée	Totale
Fonctionnement extérieur	Impossible	Oui
Adaptation jour/nuit	Fixe	Seuil dynamique configurable
Infrastructure requise	Câblage dédié + capteurs partout	Caméras IP + serveur existants
Coût de déploiement	Élevé	Faible (matériel standard)
Intégration systèmes tiers	Limitée (relais, bus propriétaires)	API REST standard (HTTP/JSON)

Conclusion

Ce premier chapitre a établi le contexte général et les enjeux du domaine de la détection d'incendie. L'analyse des statistiques mondiales confirme l'ampleur des pertes humaines et économiques liées aux incendies, justifiant pleinement l'investissement dans des solutions de détection plus performantes. L'examen des systèmes traditionnels détecteurs de fumée, thermiques, de gaz révèle des limitations structurelles importantes : fausses alarmes fréquentes, inefficacité dans les environnements complexes (grands volumes, zones ventilées, extérieurs), détection tardive et absence de localisation précise.

Ces constats motivent le recours à une approche par vision par ordinateur et apprentissage profond, capable de détecter visuellement les flammes dès leur apparition, dans des conditions variées, avec une précision et une réactivité que les systèmes conventionnels ne peuvent atteindre. Le chapitre suivant présentera les fondements théoriques de l'intelligence artificielle et des réseaux de neurones convolutifs mis en œuvre dans ce projet.

CHAPITRE II :

Outils et technologies utilisées

Chapitre II: Outils et technologies utilisés

Introduction

La conception d'un système de détection d'incendie par intelligence artificielle est indissociable du choix des outils technologiques qui le sous-tendent. En effet, la pertinence d'une solution logicielle ne réside pas uniquement dans ses algorithmes, mais dans la cohérence et la complémentarité des technologies sur lesquelles elle repose. Un outil inadapté peut pénaliser les performances globales du système, compliquer la maintenance, ou rendre difficile l'intégration avec des infrastructures existantes.

Ce chapitre présente, de manière rigoureuse et motivée, l'ensemble des outils et bibliothèques logicielles mis en œuvre dans le cadre de ce projet. Pour chaque technologie retenue, nous exposerons sa nature, son histoire, ses caractéristiques techniques, et le rôle précis qu'elle joue dans l'architecture du système développé. Les outils sont présentés selon leur niveau d'abstraction, en partant du langage de base jusqu'aux bibliothèques spécialisées.

II.1 Python : Langage de Développement

II.1.1 Présentation et Historique de Python

A. Nature et Philosophie du Langage

Python est un langage de programmation interprété, multi-paradigme et de haut niveau, dont la conception privilégie la lisibilité et la productivité du développeur. Contrairement aux langages compilés, Python exécute le code ligne par ligne, ce qui facilite le prototypage rapide et le débogage, des atouts majeurs pour le développement de systèmes complexes de vision par ordinateur.

Sa philosophie est régie par le « Zen de Python » (PEP 20), qui stipule que « la lisibilité compte » et que « la simplicité est préférable à la complexité ». Cette approche se traduit par:

- **Une syntaxe épurée :** L'utilisation de l'indentation obligatoire pour délimiter les blocs de code, éliminant ainsi le besoin d'accolades et réduisant l'encombrement visuel.
- **Un typage dynamique fort :** Les variables n'ont pas besoin de déclaration de type explicite, mais le langage interdit les opérations entre types incompatibles, garantissant ainsi une certaine sécurité lors de l'exécution.

B. Origines et Évolution Systémique

Chapitre II: Outils et technologies utilisés

Le langage a été créé par Guido van Rossum à la fin des années 1980 au Centrum Wiskunde & Informatica (CWI) aux Pays-Bas, avec une première version publique parue en 1991. Le nom « Python » ne fait pas référence au reptile, mais est un hommage à la troupe d'humour britannique Monty Python's Flying Circus.

L'évolution de Python a été marquée par deux jalons majeurs :

1. **Python 2.0 (2000)** : Introduction de fonctionnalités modernes comme les listes en compréhension et un système de gestion de la mémoire par comptage de références.
2. **Python 3.0 (2008)** : Une rupture majeure visant à corriger les défauts de conception du langage. Bien que non rétrocompatible, cette version a permis de moderniser la gestion du texte (Unicode) et d'optimiser les performances globales.

Pour le présent projet que nous allons voir dans le chapitre 3 , nous avons adopté la version Python 3.10, choisie pour sa stabilité et sa compatibilité optimale avec les bibliothèques d'intelligence artificielle utilisées.

C. Positionnement et Écosystème Actuel

Aujourd'hui, Python s'est imposé comme le leader mondial des langages de programmation selon les indices TIOBE et les rapports de Stack Overflow. Sa popularité repose sur une communauté mondiale massive qui alimente le Python Package Index (PyPI), comptant plus de 400 000 bibliothèques prêtes à l'emploi. [1][8]

Cette omniprésence est particulièrement visible dans les secteurs de pointe :

- Calcul scientifique et Data Science : Grâce à des piliers comme NumPy et Pandas.
- Intelligence Artificielle : Interface principale pour les frameworks de Deep Learning (PyTorch, TensorFlow).
- Développement d'interfaces et API : Frameworks robustes tels que PyQt et FastAPI.

Chapitre II: Outils et technologies utilisés

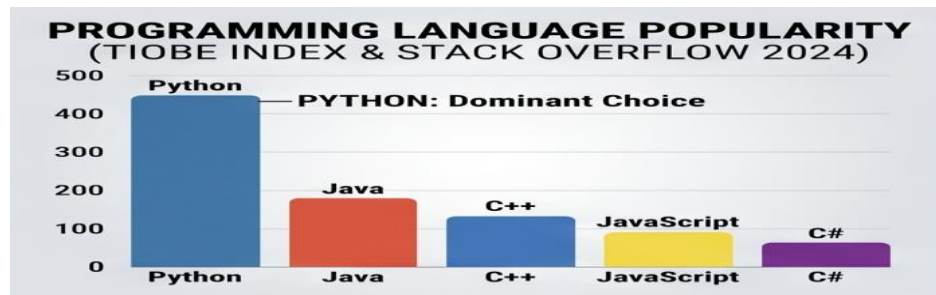


Figure II.1 Popularité des langages de programmation

Tableau II.1 Caractéristiques et leurs descriptions

Caractéristiques	Description
Interprété	Le code est exécuté ligne par ligne sans étape de compilation préalable, ce qui accélère le cycle de développement
Typage dynamique	Les types des variables sont déterminés à l'exécution, offrant souplesse et rapidité de développement
Multi-paradigme	Supporte la programmation orientée objet, fonctionnelle et procédurale dans un même programme
Portabilité	Un script Python s'exécute sans modification sur Windows, macOS et Linux
Riche bibliothèque standard	Modules intégrés pour les fichiers, réseaux, threads, JSON, expressions régulières, et bien plus
Extensibilité	Possibilité d'appeler des bibliothèques C/C++ pour les parties critiques en performance
Communauté active	Plus de 400 000 paquets disponibles sur le dépôt PyPI (Python Package Index)

II.1.2 Python et l'Intelligence Artificielle

Python est devenu le langage de référence incontesté de l'intelligence artificielle et du machine learning. Cette dominance s'explique par plusieurs facteurs convergents :

- L'ensemble des grands frameworks d'apprentissage profond PyTorch, TensorFlow, Keras disposent de leur API principale en Python
- Les bibliothèques de calcul scientifique (NumPy, SciPy, Pandas) sont natives Python et hautement optimisées
- La communauté de recherche en IA publie quasi-exclusivement ses implémentations en Python
- La syntaxe concise de Python permet de prototyper et d'expérimenter rapidement de nouvelles idées

Dans le cadre de ce projet, Python constitue la colonne vertébrale unificatrice qui permet d'orchestrer l'inférence YOLOv8, le traitement vidéo OpenCV, l'interface graphique PyQt6 et le serveur REST FastAPI au sein d'un pipeline cohérent et performant.

II.2 YOLO

II.2.1 Contexte et Motivation

Avant l'émergence de YOLO (You Only Look Once), les approches dominantes de la détection d'objets reposaient sur des architectures à deux étapes (two-stage detectors), telles que R-CNN (Regions with Convolutional Neural Networks, 2014) et ses dérivés Fast R-CNN et Faster R-CNN. Ces méthodes procèdent en deux temps : elles génèrent d'abord un ensemble de propositions de régions susceptibles de contenir un objet (region proposals), puis classifient chacune de ces régions indépendamment. Si cette approche produit des résultats précis, elle est intrinsèquement lente de quelques secondes à plusieurs dizaines de secondes par image ce qui la rend incompatible avec les applications temps réel.

C'est dans ce contexte que YOLO a été proposé en 2015 par Joseph Redmon, Santosh Divvala, Ross Girshick et Ali Farhadi. La contribution majeure de YOLO est de reformuler la détection d'objets comme un problème de régression unique, traité en un seul passage du

Chapitre II: Outils et technologies utilisés

réseau sur l'image, éliminant ainsi le coûteux processus de génération de propositions de régions [2].

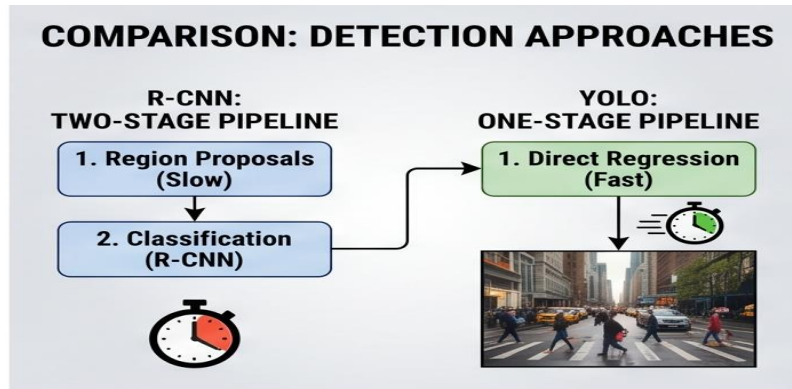


Figure II.2 Approche de détection

II.2.2 Principe Fondamental

Le principe de YOLO repose sur une idée simple : traiter la détection d'objets comme un problème de régression directe, en analysant l'image en une seule passe plutôt qu'en plusieurs étapes successives. C'est précisément cette approche qui lui confère sa vitesse de traitement exceptionnelle et qui justifie son nom : *You Only Look Once*.

Étape 1 : Division en grille

L'image d'entrée est divisée en une grille de $S \times S$ cellules égales. Chaque cellule est responsable de détecter tout objet dont le centre se trouve à l'intérieur de ses limites. Dans le cadre de FireDetect Pro, si une flamme apparaît dans la partie supérieure droite de l'image, c'est la cellule correspondante qui sera chargée de la signaler.

Étape 2 : Prédiction par cellule

Pour chaque cellule, le réseau prédit simultanément trois informations :

La boîte englobante (bounding box) : un rectangle défini par les coordonnées de son centre (x, y), sa largeur (w) et sa hauteur (h), qui encadre précisément l'objet détecté.

Chapitre II: Outils et technologies utilisés

Le score de confiance : une valeur entre 0 et 1 qui indique à la fois la probabilité qu'un objet soit présent et la précision avec laquelle la boîte l'encadre. Un score proche de 1 signifie que le modèle est très sûr de sa détection.

La probabilité de classe : la probabilité que l'objet détecté appartienne à une classe donnée dans notre cas, la classe *feu*.

Étape 3 : Calcul du score final

Le score de détection final est obtenu en combinant le score de confiance et la probabilité de classe. Plus ce score est élevé, plus la détection est fiable. Un seuil minimal est fixé en production [dans FireDetect Pro, ce seuil est réglé à 0,45] en dessous duquel la détection est ignorée afin de limiter les fausses alarmes.

Étape 4 : Suppression des doublons (NMS)

Une même flamme peut être détectée par plusieurs cellules voisines simultanément, générant des boîtes redondantes. La suppression des non-maximums (*Non-Maximum Suppression*, NMS) résout ce problème en ne conservant que la boîte ayant le score le plus élevé parmi toutes celles qui se chevauchent significativement, produisant ainsi une détection propre et unique par objet.

II.2.3 Évolution de la Famille YOLO

Tableau II.2 évolution de la famille YOLO

Version	Année	Innovation Principale
YOLOv1	2015	Concept fondateur Détection unifiée en un seul passage
YOLOv2	2016	Mécanisme d'ancres, Normalisation par lots, 9000 classes
YOLOv3	2018	Détection multi-échelle , backbone Darknet-53
YOLOv4	2020	CSPDarknet, Path Aggregation Network, Mosaic augmentation
YOLOv5	2020	Migration vers PyTorch, Facilité d'utilisation accrue
YOLOv6	2022	Optimisations spécifiques au déploiement industriel
YOLOv7	2022	Extended Efficient Layer Aggregation Networks (ELAN)
YOLOv8	2023	Architecture anchor-free, API unifiée, mAP 53.9% sur COCO

Chapitre II: Outils et technologies utilisés

II.2.4 Architecture et Innovations du YOLOv8

Le YOLOv8 a été publié par Ultralytics en janvier 2023. Il représente une refonte profonde de l'architecture [3] comme il est illustré figure II.3.

a) **Backbone Blocs C2f**: Le backbone utilise des blocs C2f (Cross Stage Partial with 2 convolutions). Les connexions résiduelles intégrées facilitent la propagation du gradient pendant l'entraînement. Le backbone extrait des représentations à trois niveaux de résolution (P3, P4, P5).

b) **Neck FPN et PAN** : Le neck agrège les caractéristiques via FPN (information sémantique) et PAN (information de localisation). Cette double agrégation permet de détecter précisément des objets de toutes tailles.

c) **Tête de Détection Anchor-Free** : YOLOv8 abandonne le mécanisme d'ancres prédéfinies pour prédire directement les coordonnées absolues. La tête est découpée en deux branches séparées pour la localisation et la classification.

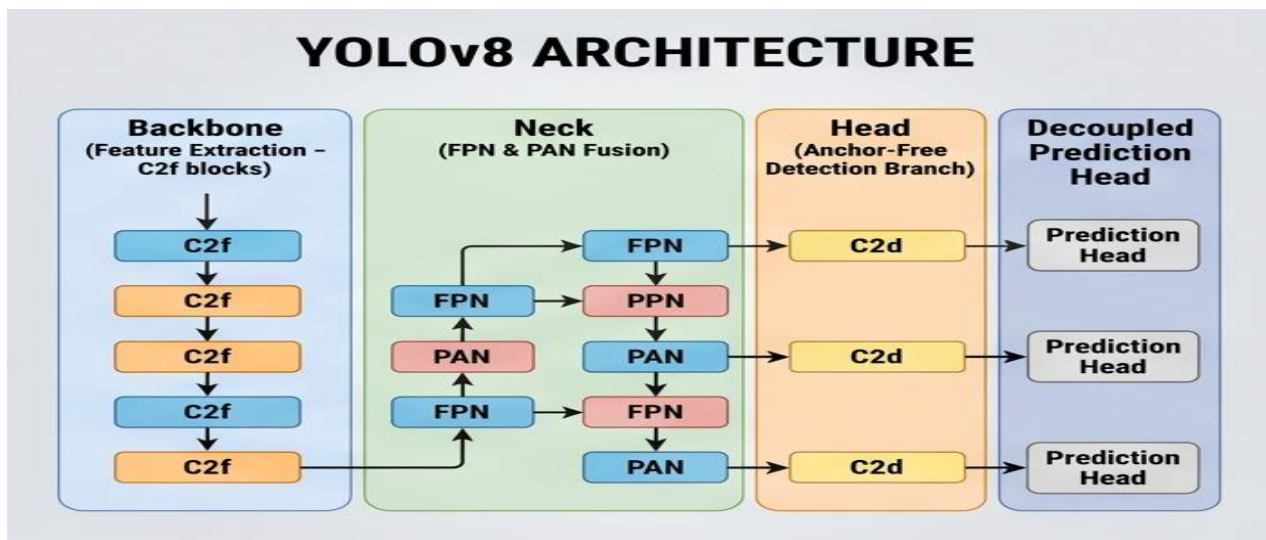


Figure II.3 l'architecture de YOLOv8

II.3 Traitement d'Image et Vision par Ordinateur (OpenCV)

Chapitre II: Outils et technologies utilisés

II.3.1 Présentation

Le OpenCV (Open Source Computer Vision Library) est une bibliothèque open source développée initialement par Intel en 1999. Elle est implémentée en C/C++ pour des performances maximales et propose plus de 2 500 algorithmes optimisés. [4]

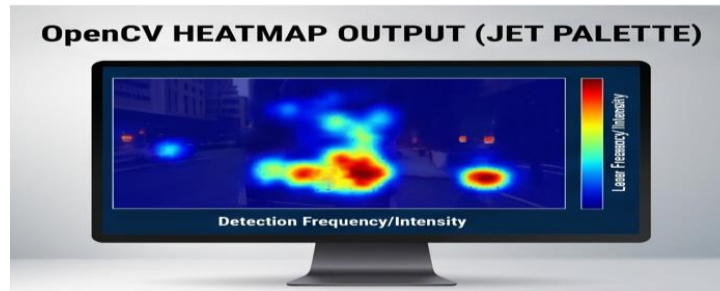


Figure II.4 Traitement d'image et vision par ordinateur

II.3.2 Rôle dans ce Projet

Tableau II.3 fonction de chaque composant du projet

Fonction	Module / Méthode	Description
Capture video	cv2.VideoCapture	Acquisition des flux depuis caméras USB et IP (RTSP)
Optimisation buffer	CAP_PROP_BUFFERSIZE	Réduction de la latence de capture à ~30 ms
Redimensionnement	cv2.resize	Adaptation des frames à 320×320 px pour l'inférence
Annotation visuelle	cv2.rectangle, putText	Tracé des boîtes englobantes et labels
Heatmap	cv2.applyColorMap	Application de la palette JET
Enregistrement	cv2.VideoWriter	Sauvegarde des clips au format MP4

Chapitre II: Outils et technologies utilisés

II.4 Framework d'Interface Graphique (PyQt6)

II.4.1 Mécanisme Central

L'un des concepts fondamentaux de framework Qt est le mécanisme de signaux et slots. Un signal est un message émis lors d'un événement, et un slot est une méthode qui y réagit. Ce mécanisme permet une architecture faiblement couplée, idéale pour le multi-thread [5].

II.5 Les Caméras Utilisées dans les Systèmes de Détection d'Incendie

Dans un système de détection d'incendie basé sur la vision par ordinateur, la caméra est le premier maillon de la chaîne de traitement. Sa qualité détermine directement la pertinence des données fournies au modèle d'intelligence artificielle, et par conséquent la précision et la fiabilité globale du système. Une caméra de faible résolution, à faible débit ou mal adaptée à l'environnement surveillé peut provoquer des faux positifs, des manques de détection ou une latence excessive. Il est donc fondamental de comprendre les différents types de caméras disponibles, leurs caractéristiques techniques, leurs performances comparatives et le contexte algérien dans lequel notre système est amené à être déployé.

Dans le cadre de notre projet, la webcam de l'ordinateur portable a été utilisée comme outil de développement et de test. Bien qu'elle permette de valider le fonctionnement du pipeline de détection, elle ne constitue pas une solution de déploiement réel dans un environnement de surveillance industrielle ou extérieure.

On distingue principalement trois catégories de caméras utilisées dans les systèmes de détection d'incendie par vision artificielle .

Chapitre II: Outils et technologies utilisés

Tableau II.4 Comparaison entre les types de camera

Critère	Caméra IP Standard	Caméra Thermique	Caméra IA-Fire (Dahua/Bosch)
Résolution	1080p – 4K	160×120 – 640×512	2MP – 5MP
Vision nocturne	IR intégré (30-50 m)	Totale (thermique)	IR intégré (20 m)
Compatibilité RTSP/OpenCV	Native	Partielle	Native
Détection nocturne incendie	Bonne	Excellente	Excellente
Usage extérieur (IP66/67)	Oui	Oui	Oui
Distance câblage	100 m (Ethernet/PoE)	100 m (Ethernet/PoE)	100 m (Ethernet/PoE)
Précision IA associée	Bonne – Très bonne	Excellente	Excellente

Chapitre II: Outils et technologies utilisés

A. La Caméra IP (Internet Protocol)

La caméra IP est une caméra numérique autonome intégrant un processeur, une mémoire et une interface réseau Ethernet (RJ45) ou Wi-Fi. Elle communique via le protocole IP et transmet son flux vidéo au format RTSP (Real Time Streaming Protocol), directement compatible avec notre système (OpenCV supporte nativement le protocole RTSP). Ses caractéristiques typiques sont : résolution de 2MP (1080p) à 8MP (4K), débit de 25 à 60 FPS, vision nocturne infrarouge intégrée jusqu'à 30-50 m, distance de câblage jusqu'à 100 m (Ethernet), indice de protection IP66/IP67, alimentation PoE (Power over Ethernet). [24]

Des études comparatives montrent que les systèmes de détection par vision artificielle atteignent des performances de précision entre 85 % et 97 % selon la qualité du flux vidéo entrant, avec des caméras IP 1080p donnant systématiquement de meilleurs résultats que les webcams standard à résolution équivalente, grâce à leur stabilité de flux, leur faible bruit d'image et leur vision nocturne intégrée.



FigureII.5Camera IP

B. La Caméra Thermique

La caméra thermique ne capte pas la lumière visible mais le rayonnement infrarouge émis par tous les objets en fonction de leur température. Elle produit une image thermique dans laquelle chaque pixel représente une température mesurée. Sa résolution thermique varie de 160×120 à 640×512 pixels, avec une plage de température de -20°C à +2000°C et une

Chapitre II: Outils et technologies utilisés

précision de $\pm 2^{\circ}\text{C}$. Elle est considérée comme la solution la plus performante pour la détection précoce d'incendie et peut identifier un point chaud bien avant l'apparition d'une flamme visible[25].

Cependant, son coût élevé et sa complexité d'intégration la réservent aux installations industrielles critiques (data centers, entrepôts, sites pétroliers).



Figure II.6 Camera thermique

C. La Caméra IP Spécialisée Détection Incendie (IA Embarquée)

Ce type de caméra, comme le modèle Dahua IPC-HDW3849H-AI-Fire ou le système Bosch AVIOTEC 8000i IR, intègre directement à bord un algorithme de détection d'incendie et de fumée. Elle combine un capteur vidéo haute résolution et un capteur de fumée physique dans le même boîtier. Par exemple, la Dahua 5MP AI-Fire propose une résolution 5MP à 25 FPS, un capteur de fumée photoélectrique intégré (couverture 30-60 m²), une détection de flamme par IA embarquée, une vision nocturne IR (20 m) et une sortie d'alarme physique [26] [10].



Figure II.7 Camera IP avec IA embarquée

II.5.1 Déploiement des Caméras de Surveillance en Algérie

L'Algérie a entamé depuis plusieurs années une politique d'expansion significative de ses infrastructures de vidéosurveillance, créant ainsi un terrain favorable au déploiement de solutions de détection d'incendie par vision artificielle. Dès 2004, la Direction Générale de la Sûreté Nationale (DGSN) a initié l'installation de caméras de surveillance sur les principales artères d'Alger. En 2018, la capitale comptait déjà 1 500 caméras en service, avec un projet d'extension à plus de 5 000 caméras modernes dotées d'applications intelligentes [27]

En 2019, le ministre de l'Intérieur a annoncé la généralisation du système de vidéosurveillance à l'ensemble des 58 wilayas du pays. [28] Cette dynamique nationale se concrétise à travers plusieurs wilayas : à Blida, en 2024, 1 710 caméras sont déployées dont 170 intelligentes, ayant permis une augmentation de 551% des cas enregistrés entre 2023 et 2024[29]; à Béjaïa, les travaux d'installation du dispositif de télésurveillance ont atteint 95% d'avancement en mars 2024; [30] à Oran, deuxième métropole du pays, un programme de modernisation de la vidéosurveillance a été engagé dans le cadre du projet Smart City, couvrant les principales artères, les zones commerciales et les infrastructures sensibles, avec des caméras IP haute définition connectées à un centre de supervision centralisé.

Ce contexte révèle que l'Algérie dispose déjà d'une infrastructure de caméras de surveillance croissante, majoritairement composée de caméras IP Ethernet. Notre système est donc

Chapitre II: Outils et technologies utilisés

directement compatible avec l'infrastructure existante, sans nécessiter de remplacement du matériel.

II.5.2 Recommandation de camera pour le déploiement Optimal du Système

Sur la base de l'analyse des types de caméras et du contexte algérien, la recommandation principale est l'utilisation d'une caméra IP 1080p (Full HD) avec vision nocturne infrarouge, alimentée en PoE(Power over Ethernet). Ce type de caméra offre une résolution suffisante pour que le modèle YOLOv8 détecte les flammes avec une précision maximale, un flux RTSP directement compatible avec le module camera_manager.py du système, une vision nocturne intégrée essentielle pour la fiabilité du système en mode nuit (complétant notre logique d'adaptation jour/nuit), et un coût accessible pour les marques Dahua, Hikvision, TP-Link, disponibles sur le marché algérien[31].

Voici les caméras recommandées par niveau de budget dans le tableau

Tableau II.5 options de caméras recommandées par niveau de budget

Budget	Modèle recommandé	Résolution	Vision nocturne
Économique	Hikvision DS-2CD1143G2-I	4MP	IR 30 m
Standard	Dahua IPC-HDW2849H-S-IL	8MP	ColorVu 30 m
Avancé	Hikvision DS-2CD2T47G2-L	4MP	ColorVu 60 m
Industriel	Dahua AI-Fire IPC-HDW3849H	5MP + détecteur fumée	IR 20 m

Dans le cadre de notre projet, la webcam de l'ordinateur portable a constitué la solution de test disponible. Elle permet de valider intégralement le pipeline de détection (acquisition → inférence → alertes → API) dans un contexte académique. Cependant, ses limitations (absence de vision nocturne, résolution modeste, non-étanchéité) font qu'elle ne représente pas la solution cible pour un déploiement opérationnel. La migration vers une caméra IP est la première évolution recommandée avant toute mise en production du système.

Chapitre II: Outils et technologies utilisés

II.5 Récapitulatif des Technologies

Tableau II.6 Caractéristique des outils

Outil	Catégorie	Version min.	Rôle
Python	Langage	3.10	Langage de développement principal
Ultralytics	IA	8.0.0	Moteur de détection YOLOv8
OpenCV	Vision	4.8.0	Capture et traitement image
PyQt6	GUI	6.6.0	Interface desktop
FastAPI	Web	0.110.0	API REST
NumPy	Calcul	1.24.0	Calcul heatmap

Conclusion

Ce chapitre a présenté de manière rigoureuse l'ensemble des outils et technologies qui constituent le socle du système développé. Le choix délibéré de technologies open source, activement maintenues et interopérables garantit la pérennité, l'évolutivité et la portabilité du système développé. Le chapitre suivant présentera la conception et l'implémentation concrète du système.

CHAPITRE III :

Conception et Réalisation et Application.

Chapitre III: Conception et Réalisation du Système

Introduction

Ce chapitre constitue le cœur technique du projet ; il détaille l'ensemble du processus de conception, de développement et de réalisation du système de détection d'incendie par intelligence artificielle. Après avoir présenté les fondements théoriques en chapitre I et les outils technologiques utilisés en chapitre II, ce chapitre décrit la manière dont ces éléments ont été assemblés pour former un système complet, fonctionnel et optimisé.

La démarche adoptée suit un processus de développement itératif structuré en cinq étapes principales :

1. la collecte et la préparation des données d'entraînement
2. l'entraînement personnalisé du modèle YOLOv8
3. la conception de l'architecture logicielle globale
4. le développement de chaque module fonctionnel
5. l'intégration et la validation du système complet.

III.1 Architecture de notre système

III.1.1 Présentation générale

Notre système est organisé en plusieurs parties qui travaillent ensemble pour assurer la détection d'incendie en temps réel. Chaque partie a un rôle précis, ce qui rend le programme plus clair, plus facile à modifier et plus simple à tester . Les différentes parties communiquent entre elles à l'aide de files d'attente et du mécanisme de signaux et slots de PyQt6. Cela permet d'éviter les dépendances directes entre les modules et de garder un fonctionnement fluide. Comme illustré dans le tableau ci-dessous, notre système se compose de plusieurs modules, chacun utilisant une technologie adaptée à sa fonction.

Chapitre III: Conception et Réalisation du Système

Tableau II.1 Architecture des modules

Partie	Module	Technologie	Rôle
Acquisition	camera_manager.py	OpenCV VideoCapture	Lecture des flux USB ou RTSP
Détection	detector.py	Ultralytics YOLOv8	Inférence et seuil adaptatif jour/nuit
Alertes	alert.py	Plyer + Pygame	Notification visuelle et sonore
Enregistrement	recorder.py	OpenCV VideoWriter	Création automatique de clips MP4 horodatés
Heatmap	heatmap.py	NumPy + OpenCV	Suivi des zones à risque récurrentes
Interface	main_window.py	PyQt6	Tableau de bord en temps réel
API REST	server.py	FastAPI + Uvicorn	Communication avec d'autres systèmes

Chapitre III: Conception et Réalisation du Système

III.1.2 Structure du Projet de Détection d'Incendie

Le projet FireDetect Pro est organisé selon une architecture modulaire, où chaque fichier ou dossier possède un rôle unique et bien défini. Cette séparation claire des responsabilités facilite la maintenance, les tests et les évolutions futures du système.

L'ensemble du projet est regroupé dans un dossier racine nommé `fire_detection_v2/`, structuré comme suit :

Les fichiers racines :

Le socle du projet, à la racine se trouvent les fichiers essentiels au démarrage et à la configuration globale. `main.py` est le point d'entrée unique : c'est lui qui est exécuté au lancement et qui orchestre tous les autres modules. `config.py` centralise tous les paramètres modifiables du système (seuils de détection, sources caméras, durées d'alerte) afin qu'un simple changement dans ce fichier suffise à reconfigurer l'ensemble de l'application. `fire_model.pt` contient les poids du modèle YOLOv8 entraîné spécifiquement sur des images de feu, et `requirements.txt` liste toutes les bibliothèques Python nécessaires à l'installation.

Le dossier `core/` :

Le cerveau du système, ce dossier regroupe toute la logique métier du projet, c'est-à-dire les traitements intelligents qui font fonctionner la détection. `detector.py` constitue le moteur central : il charge le modèle YOLOv8, analyse chaque image reçue et intègre la logique d'adaptation automatique jour/nuit. `camera_manager.py` gère la connexion et la lecture simultanée de plusieurs caméras, qu'elles soient USB branchées directement ou IP distantes via le protocole RTSP. `recorder.py` prend en charge l'enregistrement automatique de clips vidéo au format MP4 dès qu'un incendie est détecté. `heatmap.py` génère la carte thermique qui visualise les zones les plus fréquemment touchées par des détections. Enfin, `alert.py` déclenche les alertes sonores et notifications bureau, avec un mécanisme anti-spam pour éviter les répétitions excessives.

Chapitre III: Conception et Réalisation du Système

Le dossier `ui/` :

C'est l'interface utilisateur, ce dossier contient `main_window.py`, le fichier unique qui définit l'ensemble de l'interface graphique PyQt6 la fenêtre principale, les onglets caméras, la heatmap, le journal des détections et le panneau de paramètres.

Le dossier `api/` :

C'est la communication externe, `server.py` implémente le serveur FastAPI qui expose les données du système via une API REST. Cela permet à des applications tierces — tableau de bord web, application mobile, système SCADA — de consulter l'état du système et l'historique des détections à distance.

Les dossiers de sortie :

Sont les données générées comme `recordings/` stocke automatiquement les clips vidéo MP4 générés lors de chaque détection d'incendie, horodatés pour faciliter la traçabilité. `snapshots/` conserve les captures d'écran PNG prises au moment précis de chaque alerte, constituant ainsi une preuve visuelle archivée de chaque événement détecté.

Cette organisation en couches [configuration, logique métier, interface, API, sorties]reflète les bonnes pratiques de développement logiciel et garantit que chaque composant peut être modifié ou remplacé indépendamment sans impacter le reste du système.

III.1.3 Fonctionnement général en temps réel

Notre système fonctionne avec plusieurs tâches exécutées en parallèle. Pendant qu'une partie capture la vidéo, une autre réalise la détection, une autre met à jour l'interface, et une autre gère les alertes. Cette organisation permet d'éviter les blocages et de garder un affichage fluide.

Le traitement en temps réel repose donc sur une séparation claire des tâches, ce qui améliore la rapidité et la stabilité du système.

Chapitre III: Conception et Réalisation du Système

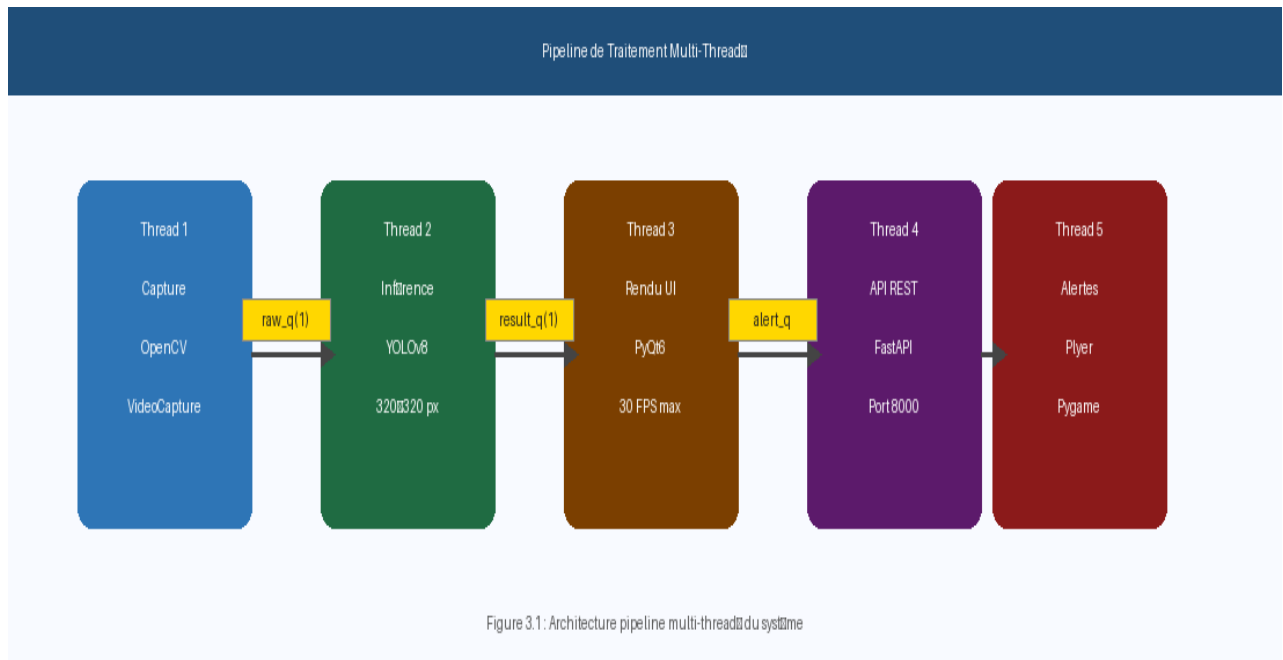


Figure III.1 pipeline multi-thrèade du système flux de données entre les 5 threads

➤ Thread 1 : capture

Outils : OpenCV / VideoCapture

C'est le point de départ. Il capture les images en continu depuis la caméra (webcam ou IP). Il envoie ensuite chaque image dans une file d'attente appelée `raw_q` comme un tapis roulant qui transporte les images brutes vers l'analyse.

➤ Thread 2 : Inférence

Outils : YOLOv8 / 320×320 px

Il reçoit les images brutes depuis `raw_q`, les redimensionne en 320×320 pixels et les passe au modèle YOLOv8 pour détecter le feu. Les résultats annotés (boîtes, scores) sont envoyés dans `result_q`.

.

➤ Thread 3 : Rendu UI

Outils : PyQt6 / 30 FPS max

Il reçoit les résultats depuis `result_q` et les affiche en temps réel dans l'interface graphique à une cadence de 30 images par seconde. Si un feu est détecté, il envoie une alerte dans `alert_q`. Thread 5 Alertes : gère les notifications desktop et le son en asynchrone, sans impacter la vitesse du flux vidéo.

➤ Thread 4 : API REST

Chapitre III: Conception et Réalisation du Système

Outils : FastAPI / Port 8000

Il tourne en parallèle et expose les données du système via une API REST sur le port 8000. Des applications externes (mobile, web, SCADA) peuvent consulter l'état du système à tout moment.

➤ Thread 5 — Alertes

Outils : Plyer / Pygame

Il reçoit les signaux d'alerte depuis alert_q et déclenche simultanément une notification bureau (Plyer) et une alarme sonore (Pygame) pour avertir l'opérateur.

III.2 Acquisition du Dataset d'Entraînement

III.2.1 Collecte et Composition des Données

La qualité des données joue un rôle essentiel dans les performances d'un modèle de détection. Pour notre projet, nous avons constitué un dataset d'environ 11 000 images issues de plusieurs sources afin d'obtenir une bonne diversité de scènes.

Ce dataset contient des images de feux de forêt, d'incendies en intérieur, de vidéos réelles, de caméras de surveillance et d'images augmentées artificiellement. Cette diversité permet d'améliorer la robustesse du modèle.

Tableau III.2 composition et sources du Dataset d'entraînement

Source	Quantité	Type	Intérêt
D3Fire / Kaggle	3 000 images	Feux de forêt	Scènes extérieures
FireNet	2 500 images	Intérieur / extérieur	Diversité des situations
Vidéos YouTube	2 000 images	Incendies réels	Angles variés
CCTV publiques	1 500 images	Surveillance	Conditions réelles de caméra
Augmentation synthétique	2 000 images	Transformations	Robustesse renforcée

Chapitre III: Conception et Réalisation du Système

III.2.2 Annotation des Données au Format YOLO

Après la collecte des images, chaque photo du dataset a été annotée manuellement. Annoter une image signifie simplement dessiner un rectangle autour de chaque flamme visible, afin d'indiquer au modèle où se trouve le feu.

Comment fonctionne l'annotation YOLO ?

Le rectangle dessiné autour de la flamme est sauvegardé dans un fichier texte .txt portant le même nom que l'image. Chaque flamme annotée correspond à une ligne dans ce fichier, contenant cinq valeurs :

0 0.512 0.438 0.324 0.276

Ces cinq valeurs ont chacune une signification précise :

0 : la classe détectée

0.512 : le centre horizontal de la flamme, situé à 51.2% de la largeur de l'image

0.438 : le centre vertical de la flamme, situé à 43.8% de la hauteur de l'image

0.324 : la largeur du rectangle, qui représente 32.4% de la largeur totale de l'image

0.276 : la hauteur du rectangle, qui représente 27.6% de la hauteur totale de l'image

Ces valeurs sont toutes comprises entre 0 et 1 car elles sont exprimées en pourcentages plutôt qu'en pixels. Cela présente un avantage important, les annotations restent valables quelle que soit la résolution de l'image.

Outils utilisés

Deux outils ont été employés pour cette étape. LabelImg est un logiciel open source qui permet de dessiner les rectangles d'annotation directement sur les images, image par image. Roboflow est une plateforme en ligne qui a servi à vérifier la qualité des annotations, corriger les erreurs éventuelles et préparer le dataset final pour l'entraînement.

À la fin de cette étape, chaque image du dataset est accompagnée de son fichier .txt contenant les coordonnées de toutes les flammes qu'elle contient. C'est à partir de ces paires image/annotation que le modèle YOLOv8 apprend à détecter le feu de manière autonome.

Chapitre III: Conception et Réalisation du Système

III.2.3 Augmentation des Données

Pour améliorer la robustesse du modèle, plusieurs transformations ont été appliquées automatiquement pendant l'entraînement. Nous avons utilisé :

le retournement horizontal

- la rotation
- les variations de luminosité
- le flou
- le bruit
- les techniques natives de YOLO comme Mosaic et MixUp.

Ces augmentations permettent au modèle de mieux généraliser et de rester performant dans des conditions différentes.

III.2.4 Répartition des Dataset

Le dataset a été réparti en trois parties :

- 80% pour l'entraînement,
- 10% pour la validation,
- 10% pour le test.

Cette répartition permet d'entraîner le modèle tout en gardant une partie des données pour vérifier ses performances réelles.

III.3 Entraînement du Modèle YOLOv8

III.3.1 Apprentissage par transfert

Au lieu de partir de zéro, nous avons utilisé un modèle YOLOv8n déjà pré-entraîné sur le dataset COCO. Cette méthode, appelée apprentissage par transfert, permet de gagner du temps et d'obtenir de meilleures performances avec un dataset limité.. [3][11]

Cette approche présente deux avantages majeurs : le temps d'entraînement est réduit de plusieurs jours à quelques heures, et les performances sur un dataset de taille limitée sont bien supérieures à celles d'un entraînement from scratch.

Chapitre III: Conception et Réalisation du Système

III.3.2 paramètres d'Entraînement

La commande d'entraînement YOLOv8 est exécutée avec les hyper paramètres suivants, soigneusement calibrés pour équilibrer vitesse de convergence, performances et généralisation

Tableau III.3 les paramètres d'entrainement et leur justifications

Hyperparamètre	Valeur	Justification
Modèle de base	yolov8n.pt	Nano — léger, rapide sur CPU
Epochs	100	Équilibre apprentissage / sur-apprentissage
Batch size	16	Adapté à 8 Go VRAM
Image size	640×640 px	Taille standard YOLOv8
Learning rate initial	0.01	Valeur par défaut YOLOv8
Learning rate final	0.0001	Décroissance cosinus
Optimiseur	SGD mom. 0.937	Robuste pour la détection
Weight decay	0.0005	Régularisation L2 anti-overfitting
Warmup epochs	3	Évite la divergence initiale
Seuil NMS (IoU)	0.45	Suppression des détections redondantes

III.3.3 Résultats obtenus

Les résultats montrent que le modèle atteint de bonnes performances. La précision est d'environ 92%, le rappel d'environ 88% et le mAP@0.5 d'environ 91%. Ces résultats confirment que le modèle est efficace pour détecter les flammes en temps réel comme illustré ci-dessous :

Chapitre III: Conception et Réalisation du Système

Tableau III.4 Métriques de performance évaluées sur l'ensemble de test

Métrique	Valeur obtenue	Interprétation
Precision (P)	~92 %	92 % des détections positives sont réelles
Recall (R)	~88 %	88 % des feux présents sont correctement détectés
mAP@0.5	~91 %	Excellente localisation au seuil IoU 50 %
mAP@0.5:0.95	~68 %	Bonne généralisation multi-seuils IoU
Vitesse CPU	~35 ms/image	28 FPS effectifs sans GPU temps réel garanti
Vitesse GPU	~8 ms/image	60+ FPS avec GPU fluidité maximale
Taille modèle .pt	~6 Mo	Déploiement embarqué possible (Jetson, Pi 5)

III.4 Adaptation Dynamique Jour/Nuit

FireDetect Pro ajuste automatiquement sa sensibilité de détection selon la luminosité de la scène, sans intervention manuelle.

Comment ça fonctionne ?

À chaque image capturée, le système mesure la luminosité moyenne en niveaux de gris :

python

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Chapitre III: Conception et Réalisation du Système

```
brightness = gray.mean()
```

`cv2.cvtColor` convertit l'image en niveaux de gris. `gray.mean()` calcule la luminosité moyenne de tous les pixels, entre **0** (noir total) et **255** (blanc total).

Ensuite le système choisit automatiquement le bon seuil :

Python

```
if brightness > 80:
    conf_threshold = 0.50
    mode = 'JOUR'
else:
    conf_threshold = 0.35
    mode = 'NUIT'
```

Si la luminosité dépasse **80** → mode **JOUR**, seuil à **50%** : le modèle est strict pour éviter les fausses alarmes causées par les reflets ou lumières vives.

Si la luminosité est en dessous de **80** → mode **NUIT**, seuil à **35%** : le modèle est plus sensible pour ne rater aucune flamme dans l'obscurité.

Enfin, la détection est lancée avec le seuil choisi :

Python

```
results = model(frame_resized, conf=conf_threshold,
verbose=False)
```

`model()` appelle YOLOv8 sur l'image. `conf=conf_threshold` lui impose le seuil adapté. `verbose=False` désactive les messages inutiles dans la console.

III.5 Développement des Modules Fonctionnels

Chapitre III: Conception et Réalisation du Système

III.5.1 Gestion des Caméras

Le module CameraManager permet de récupérer les flux vidéo depuis des caméras USB ou IP. Chaque caméra fonctionne dans un thread séparé, ce qui évite les blocages et permet de traiter plusieurs flux en même temps.

Connexion à une caméra USB

```
cap = cv2.VideoCapture(0) # 0 = première webcam
```

Connexion à une caméra IP via RTSP

```
cap = cv2.VideoCapture('rtsp://admin:pwd@192.168.1.100:554/stream')
```

Optimisation critique : réduction du buffer de capture

```
cap.set(cv2.CAP_PROP_BUFFERSIZE, 1) # Latence -30 ms
```

III.5.2 Module de détection et moteur d'Inférence

La classe FireDetector est le cœur du système. Elle charge le modèle YOLOv8 et analyse chaque image pour détecter la présence de feu.

Initialisation

```
python

def __init__(self, model_path, day_conf=0.5, night_conf=0.35):

    self.model = YOLO(model_path)

    self.day_conf = day_conf

    self.night_conf = night_conf
```

Chapitre III: Conception et Réalisation du Système

Au démarrage, le système charge le fichier modèle .pt en mémoire et mémorise les deux seuils de confiance : 50% pour le jour et 35% pour la nuit.

Étape 1 : Choix du seuil jour/nuit

python

```
brightness = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY).mean()

conf = self.day_conf if brightness > 80 else self.night_conf
```

Le système mesure la luminosité moyenne de l'image. Si elle dépasse 80, il applique le seuil jour. Sinon, il applique le seuil nuit. Ce choix se fait automatiquement à chaque image.

Étape 2 : Redimensionnement

python

```
small = cv2.resize(frame, (320, 320))
```

L'image est réduite à 320×320 pixels avant l'analyse. Cela réduit considérablement la charge de calcul sur le CPU tout en conservant une précision suffisante pour détecter les flammes.

Étape 3 : Inférence YOLOv8

python

```
results = self.model(small, conf=conf, verbose=False)
```

Chapitre III: Conception et Réalisation du Système

L'image réduite est envoyée au modèle YOLOv8. Celui-ci analyse l'image et retourne la liste de toutes les flammes détectées avec un score supérieur au seuil choisi. `verbose=False` évite l'affichage de messages inutiles dans la console.

Étape 4 : Extraction et localisation

python

```
for box in results[0].boxes:
    x1, y1, x2, y2 = box.xyxy[0]
    zone = self._get_zone(x1, y1, x2, y2)
    detections.append({'bbox': (x1, y1, x2, y2),
                      'conf': float(box.conf[0]),
                      'zone': zone})
```

Pour chaque flamme détectée, le système récupère trois informations : les coordonnées du rectangle qui l'encadre (`x1, y1, x2, y2`), le score de confiance indiquant la certitude de la détection, et la zone dans laquelle elle se trouve selon une grille 3×3 (par exemple : Centre, Haut-Gauche, Bas-Droite). Ces informations sont retournées sous forme de liste pour être affichées dans l'interface.

III.5.3 Localisation par Grille 3×3

Le système détermine automatiquement dans quelle zone du champ de vision se situe l'incendie détecté. L'image est divisée en une grille de 9 zones (Haut-Gauche, Haut-Centre, Haut-Droite, Centre-Gauche, Centre, Centre-Droite, Bas-Gauche, Bas-Centre, Bas-Droite). La zone est calculée d'après le centre de la boîte englobante détectée.

Chapitre III: Conception et Réalisation du Système

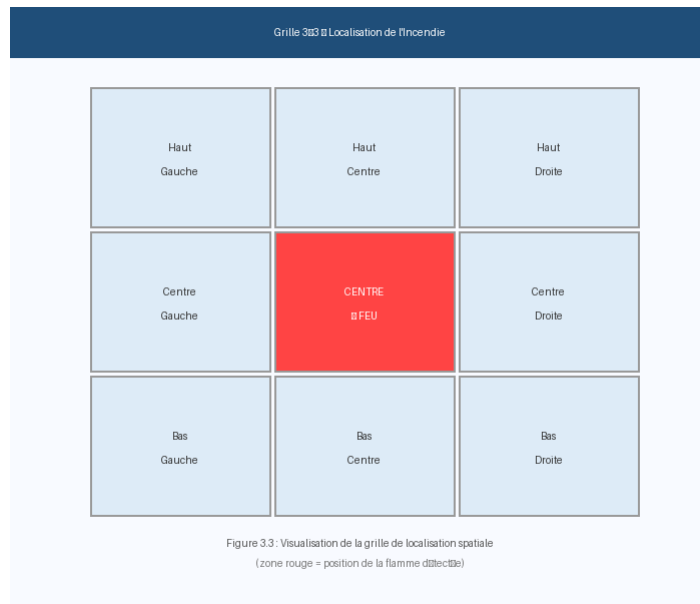


Figure III.2GRILE 3x3

III.5.4 Carte thermique des détections

La heatmap est une visualisation qui montre les zones de l'image où le feu apparaît le plus fréquemment. Plus une zone est détectée souvent, plus elle apparaît en rouge vif. Les zones rarement touchées restent en bleu. Cela permet d'identifier rapidement les endroits les plus à risque sur le site surveillé.

Initialisation

```
pythondef __init__(self, shape, decay=0.95):  
    self.map = np.zeros(shape, dtype=np.float32)  
    self.decay = decay
```

Au démarrage, le système crée une grille vide de la même taille que l'image, remplie de zéros. Chaque case de cette grille correspond à un pixel de la vidéo. $\text{decay}=0.95$ est le coefficient d'oubli progressif : les anciennes détections s'effacent lentement pour que la carte reste représentative de la situation actuelle.

Mise à jour à chaque détection

```
pythondef update(self, detections):  
    self.map *= self.decay
```

Chapitre III: Conception et Réalisation du Système

```
for det in detections:
    x1,y1,x2,y2 = det['bbox']
    self.map[int(y1):int(y2), int(x1):int(x2)] += 1.0
```

À chaque nouvelle image, deux choses se passent. D'abord, toute la carte est multipliée par 0.95, ce qui diminue légèrement les valeurs anciennes c'est l'effet mémoire qui efface progressivement le passé. Ensuite, pour chaque flamme détectée, la zone correspondante dans la grille reçoit +1.0, ce qui l'intensifie sur la carte. Avec un decay de 0.95, le système garde en mémoire l'équivalent d'environ 20 détections récentes.

Rendu visuel superposé

```
pythondef render(self, frame):
    norm = cv2.normalize(self.map, None, 0, 255,
cv2.NORM_MINMAX)
    colored = cv2.applyColorMap(norm.astype(np.uint8),
cv2.COLORMAP_JET)
    return cv2.addWeighted(frame, 0.6, colored, 0.4, 0)
```

Pour afficher la heatmap, le système réalise trois opérations. `cv2.normalize` ramène toutes les valeurs de la grille entre 0 et 255 pour les rendre affichables. `cv2.applyColorMap` applique la palette de couleurs JET qui colore les zones froides en bleu et les zones chaudes en rouge. `cv2.addWeighted` superpose la heatmap colorée sur l'image vidéo réelle avec un mélange 60% vidéo / 40% heatmap, ce qui permet de voir simultanément la scène et les zones à risque.

III.5.5 Système d'Alertes

Le module `AlertManager` envoie une notification visuelle et sonore lorsqu'un incendie est détecté. Un mécanisme anti-spam empêche l'envoi répété d'alertes pour un même événement.

Chapitre III: Conception et Réalisation du Système

III.5.6 Enregistrement Vidéo

Le module `RecordingManager` gère l'enregistrement automatique de clips vidéo au format MP4 dès qu'une flamme est détectée, sans aucune intervention de l'utilisateur.

Comportement général

Le système démarre l'enregistrement automatiquement à la **première détection** de feu et continue tant que des flammes sont visibles. Il s'arrête après un délai configurable sans détection, évitant ainsi de créer des dizaines de fichiers courts pour un même incident.

Nomenclature horodatée

python

```
from datetime import datetime
```

```
filename =  
f'recordings/fire_{datetime.now().strftime("%Y%m%d_%H%M%S")}.m  
p4'
```

Chaque fichier reçoit automatiquement un nom basé sur la date et l'heure exacte de la détection. Par exemple, une détection le 1er décembre 2024 à 14h30 et 22 secondes génère le fichier :

```
recordings/fire_20241201_143022.mp4
```

Ce nommage automatique garantit qu'aucun fichier n'écrase un autre et permet de retrouver facilement l'enregistrement correspondant à une alerte précise dans le journal des détections.

Initialisation de l'enregistrement

python

```
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
```

Chapitre III: Conception et Réalisation du Système

```
writer = cv2.VideoWriter(filename, fourcc, fps, (width, height))
```

`cv2.VideoWriter_fourcc(*'mp4v')` définit le codec de compression vidéo au format **MP4**. `cv2.VideoWriter` crée le fichier vidéo avec le nom horodaté, le codec choisi, la cadence en images par seconde (`fps`) et la résolution (`width`, `height`) identiques à celles de la caméra source. Une fois créé, chaque image contenant une détection est écrite dans ce fichier jusqu'à l'arrêt de l'enregistrement.

III.6 Interface graphique

L'interface de notre système a été développée avec PyQt6. Elle permet de visualiser les flux vidéo en direct, les zones de détection, l'état global du système et les paramètres de configuration.

III.6.1 Architecture de l'Interface

L'interface graphique est organisée en onglets afin de regrouper toutes les fonctionnalités dans une seule fenêtre principale. La communication entre les threads de traitement et l'interface repose sur le mécanisme de signaux et slots de Qt, ce qui garantit des mises à jour sûres et fluides dans le thread principal.

L'affichage intègre plusieurs éléments :

- des boîtes englobantes colorées
- le label de détection, le pourcentage de confiance et la zone correspondant à la grille 3×3
- un badge indiquant le mode jour ou nuit avec la luminosité mesurée en temps réel
- un bouton d'enregistrement par caméra, affiché en rouge lorsque l'enregistrement est actif
- un bouton de capture instantanée pour générer un snapshot PNG

Chapitre III: Conception et Réalisation du Système

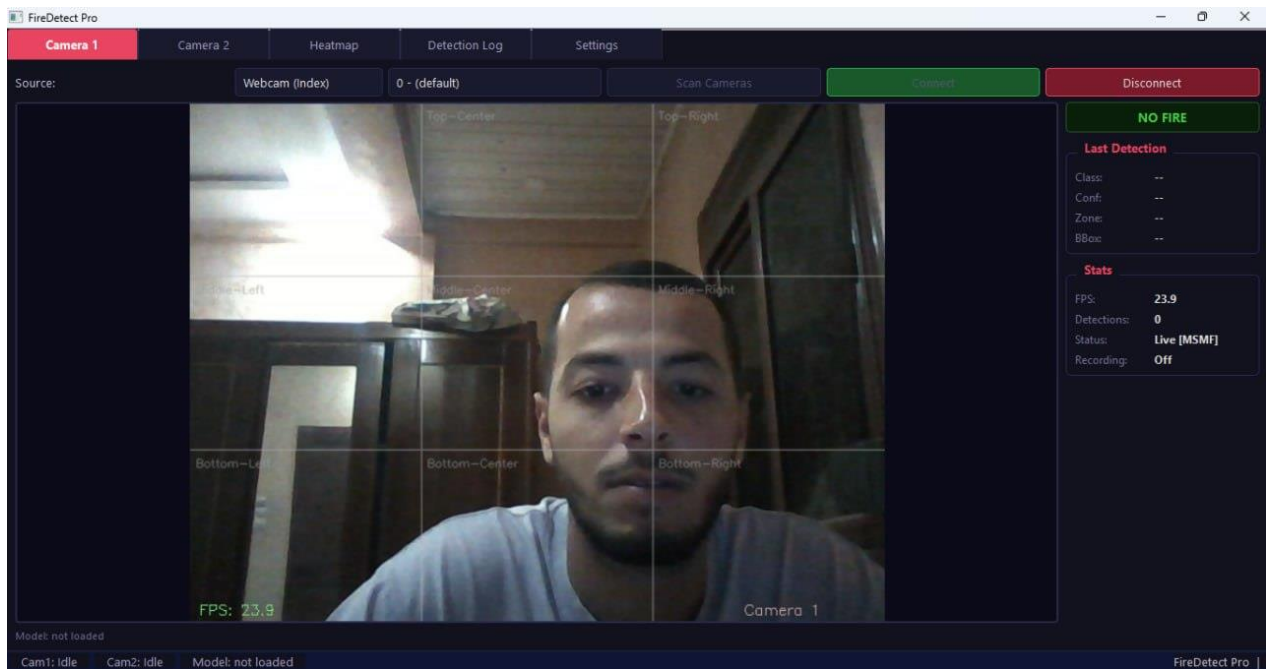


Figure III.3 Interface Graphique

III.6.2 Sidebar : État Global du Système

La barre latérale gauche centralise l'état global du système en un coup d'œil :

- ✓ Badge statut VERT (système normal) ou ROUGE animé (incendie détecté)
- ✓ Barre de confiance indiquant le niveau de certitude de la dernière détection
- ✓ Grille 3×3 de localisation visuelle de la zone de l'incendie
- ✓ Statistiques en temps réel : FPS, compteur de détections, clips enregistrés

III.6.3 Onglet Heatmap

L'onglet Heatmap présente une cartographie thermique des détections accumulées au fil du temps. Cette visualisation utilise la palette JET, où les zones bleues correspondent aux détections rares et les zones rouges aux détections fréquentes.

Une heatmap indépendante est maintenue pour chaque caméra afin de distinguer les zones de détection selon la source vidéo. Le rendu est calculé uniquement lorsque l'onglet est actif, ce qui permet de réduire la charge processeur et d'optimiser les performances globales du système.

III.6.4 Onglet de configuration avancée

Toutes les options du système sont ajustables en temps réel depuis cet onglet, sans redémarrage ni édition manuelle de fichiers de configuration :

Chapitre III: Conception et Réalisation du Système

- Chargement d'un nouveau modèle .pt (sans redémarrage)
- Curseurs pour les seuils de confiance JOUR et NUIT (1-99 %)
- Point de bascule de luminosité jour/nuite (0-255)
- Facteur de déclin de la heatmap (0.900-0.999)
- Cooldown des alertes et activation de l'enregistrement automatique

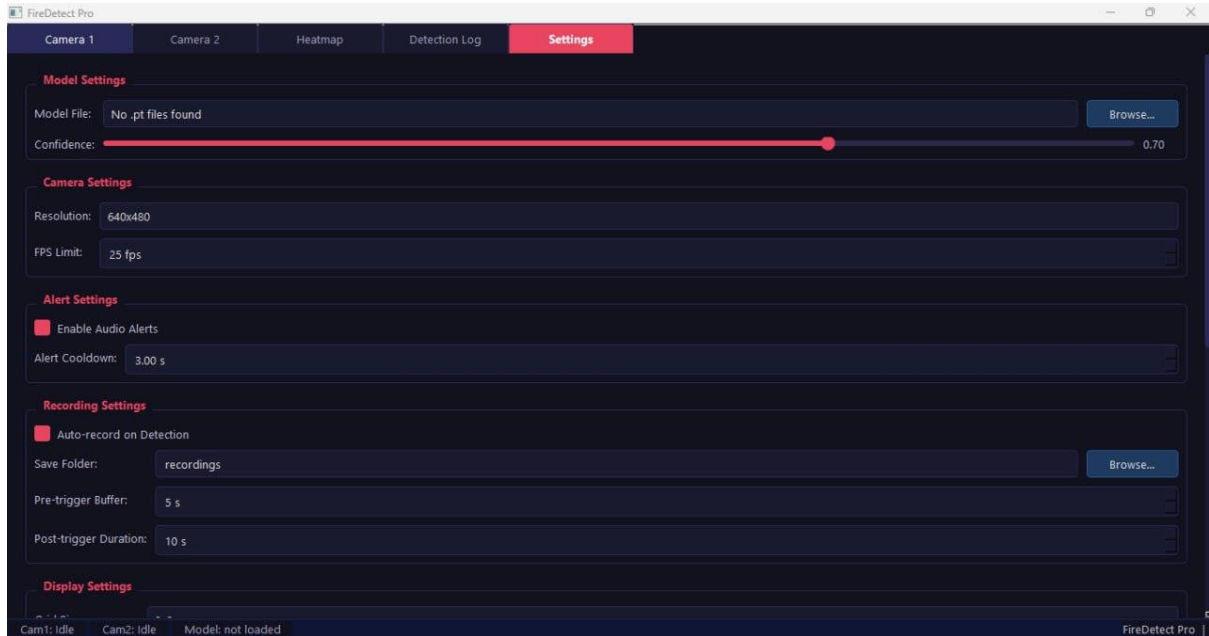


Figure III.4 Onglets parametres

III.7 API REST

L'API REST a été développée avec FastAPI et fonctionne en arrière-plan dès le lancement du système. Elle permet d'intégrer notre solution avec d'autres outils comme des tableaux de bord web, des systèmes SCADA, des BMS ou encore des plateformes IoT

III.7.1 Conception

L'API REST est implémentée avec FastAPI et tourne en arrière-plan dès le démarrage dans un thread démon. Elle permet l'intégration du système avec des outils tiers : tableaux de bord web, systèmes SCADA, BMS (Building Management Systems) ou plateformes IoT. [6]

Le tableau ci-dessous présente les principaux points d'accès, leur méthode, leur rôle ainsi qu'un exemple de réponse :

Chapitre III: Conception et Réalisation du Système

Tableau III.5 Endpoints de l'API REST et exemples de réponses JSON

Endpoint	Méthode	Description	Exemple de réponse
status	GET	État complet : détection, confiance, FPS, enregistrement	{ "fire_detected": true, "confidence": 0.87, "fps": 28.3 }
clips	GET	Liste des clips vidéo enregistrés avec leurs métadonnées	{ "clips": ["fire_20241201.mp4"], "count": 5 }
health	GET	Vérification de vie du service (healthcheck)	{ "status": "ok", "ts": 1234567890 }

III.8 Optimisations de Performance

Un effort particulier a été consacré à l'optimisation des performances pour permettre au système de fonctionner en temps réel même sur du matériel grand public sans GPU dédié. L'effet combiné de ces optimisations permet d'atteindre une cadence stable de 25-30 FPS sur un processeur Intel Core i5 de 10e génération :

Les principales optimisations appliquées sont résumées dans le tableau suivant

Chapitre III: Conception et Réalisation du Système

Tableau III.6 Optimisations de performance et leur impact mesuré

Optimisation	Implémentation	Impact mesuré
Queue maxsize=1	raw_q(1) entre threads	Aucune frame obsolète, latence minimale
Buffer OpenCV = 1	CAP_PROP_BUFFERSIZE = 1	Réduction de 30 ms de latence
Frame skipping	Inférence 1 frame / 3	Trois fois moins de calcul YOLO
Redimensionnement 320 px	cv2.resize avant inférence	Quatre fois plus rapide qu'en 640 px
Heatmap throttle	Mise à jour 1 / 5 inférences	Réduction de 80% du calcul heatmap
UI rate-limitée	Max 30 FPS affichés	Interface plus fluide
Lazy rendering heatmap	Rendu seulement si onglet actif	Aucun calcul inutile en arrière-plan

III.9 Installation et Déploiement

Le système a été conçu pour être installé facilement sur plusieurs systèmes d'exploitation.

III.9.1 Les prérequis du Système

Les éléments essentiels pour la conception

- Python 3.10 ou supérieur
- Une caméra USB ou une caméra IP accessible en RTSP

Chapitre III: Conception et Réalisation du Système

- Le fichier de poids du modèle YOLO personnalisé « fire_model.pt »
- GPU optionnel , recommandé pour les déploiements à haute cadence ou avec plusieurs cameras

III.9.2 Procédure d'Installation

L'installation de FireDetect Pro se fait en cinq étapes simples depuis un terminal, sans outil supplémentaire.

Étape 1 Créer un environnement virtuel

```
python
python -m venv venv
source venv/bin/activate      # Linux / macOS
venv\Scripts\activate        # Windows
```

Un environnement virtuel est un dossier isolé qui contient une version de Python réservée uniquement à ce projet. Cela évite tout conflit avec d'autres projets Python installés sur la machine. `activate` permet d'entrer dans cet environnement avant toute opération.

Étape 2 Installer les dépendances

```
python
pip install -r requirements.txt
```

Cette commande lit le fichier `requirements.txt` et installe automatiquement toutes les bibliothèques nécessaires : PyQt6, OpenCV, Ultralytics, NumPy et FastAPI. Une seule commande suffit pour tout installer.

Étape 3 — Placer le modèle

```
fire_detection_v2/
    fire_model.pt    copier le fichier ici
```

Chapitre III: Conception et Réalisation du Système

Le fichier `fire_model.pt` contenant les poids du modèle YOLOv8 entraîné doit être copié à la racine du projet. C'est le fichier que le système chargera au démarrage pour effectuer la détection.

Étape 4 — Vérifier la configuration

```
python
```

```
MODEL_PATH = 'fire_model.pt'
```

```
API_PORT    = 8000
```

Le fichier `config.py` centralise les paramètres principaux. Il suffit de vérifier que `MODEL_PATH` pointe bien vers le fichier modèle et que le port **8000** est disponible sur la machine.

Étape 5 — Lancer l'application

```
python
```

```
python main.py
```

Cette commande démarre l'application complète l'interface graphique PyQt6 s'ouvre, les caméras se connectent et le serveur API REST devient accessible à l'adresse :

```
http://localhost:8000/docs
```

Cette page affiche automatiquement la documentation interactive de l'API, permettant de tester tous les endpoints directement depuis un navigateur.

III.9.3 Compatibilité Multi-Plateformes

Le système est compatible avec Windows, Linux et macOS. Les bibliothèques utilisées assurent une bonne portabilité sans nécessiter de modifications importantes.

III.10 Vue d'Ensemble de l'architecture du système de notification

Chapitre III: Conception et Réalisation du Système

Le système de notification fonctionne en quatre couches successives, chacune ayant un rôle précis. L'information circule automatiquement de la caméra jusqu'à l'envoi de l'e-mail d'alerte.

Couche 1 — Traitement Vidéo

CameraThread signal frame_ready CameraTab

Le thread de capture lit en continu les images de la caméra. À chaque nouvelle image disponible, il envoie un signal `frame_ready` à l'interface graphique (CameraTab), qui appelle automatiquement la fonction `_on_frame()` pour traiter l'image.

Couche 2 — Suivi de la Durée du Feu

`_fire_start_time` `fire_duration` `_email_sent`

Dès qu'une flamme est détectée, le système enregistre l'heure de début `_fire_start_time` et calcule en permanence la durée `fire_duration` depuis la première détection. La variable `_email_sent` empêche l'envoi de plusieurs e-mails pour un même incident. Lorsque la durée dépasse le seuil configuré (par défaut **3 secondes**) le système passe à la couche suivante.

Couche 3 — Annotation de l'Instantané

`_build_email_snapshot()` → image JPEG annotée

La fonction `_build_email_snapshot()` capture l'image courante et y ajoute automatiquement les informations utiles : les boîtes de détection en rouge, le score de confiance, la zone de localisation et l'horodatage. Cette image annotée sera jointe à l'e-mail d'alerte comme preuve visuelle de l'incident.

Couche 4 — Livraison SMTP

MIMEMultipart STARTTLS Gmail SMTP

L'e-mail est construit avec MIMEMultipart, un format standard qui permet d'assembler un message contenant à la fois du texte et une pièce jointe image. La connexion au serveur Gmail se fait via le protocole STARTTLS qui chiffre les communications pour sécuriser l'envoi. L'ensemble de cette opération s'exécute dans un fil d'arrière-plan pour ne pas bloquer l'interface graphique pendant l'envoi.

III.11 Suivi de la Durée du Feu

Un déclenchement basé sur une image unique risquerait de produire de nombreuses fausses alertes. Pour réduire ce problème, le système exige que le feu soit détecté de manière continue pendant une durée définie avant d'autoriser l'envoi d'un e-mail.

Le suivi temporel du feu repose sur plusieurs variables internes. Le tableau ci-dessous résume leur rôle dans le système.

Tableau III.7 variables d'instance

Variable	Type	Rôle
<i>fire_start_time</i>	float None	Horodatage de la première détection de l'événement en cours
<i>email_threshold</i>	Float	Durée minimale avant l'envoi de l'alerte
<i>email_sent</i>	Bool	Indique si l'e-mail a déjà été envoyé
<i>last_zone</i>	Str	Dernière zone détectée avec la meilleure confiance
<i>last_conf</i>	Float	Dernier score de confiance significatif

III.11.3 Logique par Image

La logique de suivi est exécutée dans *CameraTab._on_frame()*, invoqué à chaque image émise par le fil *CameraThread* via le signal Qt *frame_ready* (cadence nominale : 25 Hz)

```
# Appelé une fois par image (~25 Hz)
if fire_dets:
    if fire_start_time is None:
        fire_start_time = time.monotonic()
        email_sent = False

    fire_duration = time.monotonic() - fire_start_time

    if not email_sent and fire_duration >= email_threshold:
        snapshot = build_email_snapshot(annotated, zone, conf, fire_duration)
        emailer.send_alert(camera_label, zone, conf, fire_duration, snapshot)
        email_sent = True
    else:
        fire_start_time = None
        email_sent = False
```

Figure III.5 : Algorithme de déclenchement de l'alerte e-mail basé sur la durée de détection

Le minuteur se réinitialise à *None* dès qu'une image sans feu est traitée, garantissant que toute réapparition constitue un nouvel événement avec son propre cycle de minuterie.

III.12 Annotation de L'image

L'image envoyée par e-mail n'est pas conservée sous sa forme brute. Elle est enrichie visuellement afin de faciliter la compréhension immédiate de l'événement.

III.12.1 Objectif

L'instantané est construit à partir d'une image déjà annotée avec les boîtes de détection, la grille de localisation et les informations de base. À cela s'ajoutent plusieurs éléments visuels qui rendent l'alerte plus claire pour le destinataire.

III.12.2 Pipeline d'Annotation en Trois parties

Le tableau suivant présente les trois couches principales utilisées dans l'annotation de l'instantané.

Chapitre III: Conception et Réalisation du Système

Tableau III.8 pipeline d'annotation en trois couches

Couche	Nom	Description
Couche 1	Bandeau Rouge (52 px)	Rectangle rouge en haut avec le message d'alerte en blanc
Couche 2	Barre d'Info (58 px)	Fond sombre en bas avec les informations de caméra, zone, confiance, durée et horodatage
Couche 3	Bordure Rouge	Bordure extérieure rouge pour renforcer la visibilité de l'alerte

```
def build_email_snapshot(self, annotated, zone, confidence, duration):
    snap = annotated.copy()          # ne pas modifier l'image en cours
    h, w = snap.shape[:2]

    # Couche 1 - Bandeau
    cv2.rectangle(snap, (0, 0), (w, 52), (0, 0, 180), cv2.FILLED)
    cv2.putText(snap, '!! FIRE DETECTED !!', (12, 34), ...)

    # Couche 2 - Barre d'information
    cv2.rectangle(snap, (0, h-58), (w, h), (20, 20, 20), cv2.FILLED)
    cv2.putText(snap, f'Camera: {label} Zone: {zone} Conf: ...', ...)
    cv2.putText(snap, f'Duration: {duration:.1f}s Time: {now str}', ...)

    # Couche 3 - Bordure
    cv2.rectangle(snap, (0, 0), (w-1, h-1), (0, 0, 220), 3)
    return snap
```

Figure III.6 Fonction `_build_email_snapshot()` construction de l'image annotée en trois couches graphiques

III.13 Module de notification par e-mail

Ce module regroupe toute la logique liée à l'envoi des notifications par e-mail.

III.13.1 Conception de la Classe

La classe EmailNotifier encapsule le paramétrage SMTP et les fonctions nécessaires à l'envoi des messages. Elle propose deux méthodes principales, résumées dans le tableau suivant :

Tableau III.9 conception de la classe EMAILNOTIFER

Méthode	Bloquante	Description
---------	-----------	-------------

Chapitre III: Conception et Réalisation du Système

Méthode	Bloquante	Description
send_alert()	Non	Envoi asynchrone d'une alerte e-mail
send_test()	Oui	Envoi synchrone pour tester la configuration

III.13.2 Mécanisme de Sécurité des Fils d'Exécution

Le module utilise un verrou (`threading.Lock`) pour empêcher plusieurs envois simultanés. Cette précaution permet d'éviter les conflits et de garantir un fonctionnement stable:

```
with self. lock:
    if self. busy:
        print('[Email] Skipped - previous send still in progress')
        return
    self. busy = True

try:
    self. connect and send(subject, body, frame)
    print(f'[Email] Alert sent to {self.recipient}')
except Exception as e:
    print(f'[Email] Failed: {e}')
finally:
    with self. lock:
        self. busy = False
```

Figure III.7 : Mécanisme de verrouillage `threading.Lock` pour la prévention des envois simultanés

III.13.3 Validation Avant Envoi

Avant de lancer le fil de livraison, `send_alert()` valide chaque paramètre requis. Si une information importante manque, l'opération est annulée afin d'éviter un échec inutile.

III.13.4 Classification de la Sévérité

Le contenu du message peut inclure un niveau de sévérité basé sur la confiance de la détection. Le tableau suivant résume cette classification.

Tableau III.10 classification de la sévérité

Seuil de Confiance	Niveau de Sévérité
--------------------	--------------------

Chapitre III: Conception et Réalisation du Système

Seuil de Confiance	Niveau de Sévérité
Très élevée	CRITIQUE
Confiance $\geq$ 65%	ÉLEVÉE
Confiance $<$ 65%	MOYENNE

III.13.5 Envoi de l'email avec pièce jointe

L'instantané annoté est encodé en JPEG à 90 % de qualité via OpenCV, puis joint en tant qu'objet image MIME. Le transport est établi sur le port TCP 587 avec négociation STARTTLS avant l'échange des identifiants :

```
ok, buf = cv2.imencode('.jpg', frame, [cv2.IMWRITE_JPEG_QUALITY, 90])
if ok:
    img = MIMEImage(buf.tobytes(), name='fire_snapshot.jpg')
    img.add_header('Content-Disposition', 'attachment',
                  filename='fire_snapshot.jpg')
    msg.attach(img)

with smtplib.SMTP(self.smtp_host, self.smtp_port, timeout=15) as srv:
    srv.ehlo()
    srv.starttls()
    srv.login(self.sender, self.password)
    srv.send_message(msg)
```

Figure III.8 : Implémentation de la livraison SMTP sécurisée avec chiffrement STARTTLS

III.14 Configuration et Intégration des Paramètres

Cette section présente les paramètres que l'utilisateur peut ajuster dans le système.

III.14.1 Paramètres Configurables par l'Utilisateur

Tableau III.11 paramètres configurables par l'utilisateur

Contrôle UI	Paramètre	Valeur par Défaut
Case à cocher	activation	Activé
Curseur de durée	email_duration	3.0 secondes
Champ expéditeur	email_sender	hrsamir211@gmail.com
Champ mot de passe	email_password	mot de passe d'application
Champ destinataire	email_receiver	Bourasamine529@gmail.com
Champ hôte SMTP	email_smtp_host	smtp.gmail.com
Curseur port SMTP	email_smtp_port	587

Chapitre III: Conception et Réalisation du Système

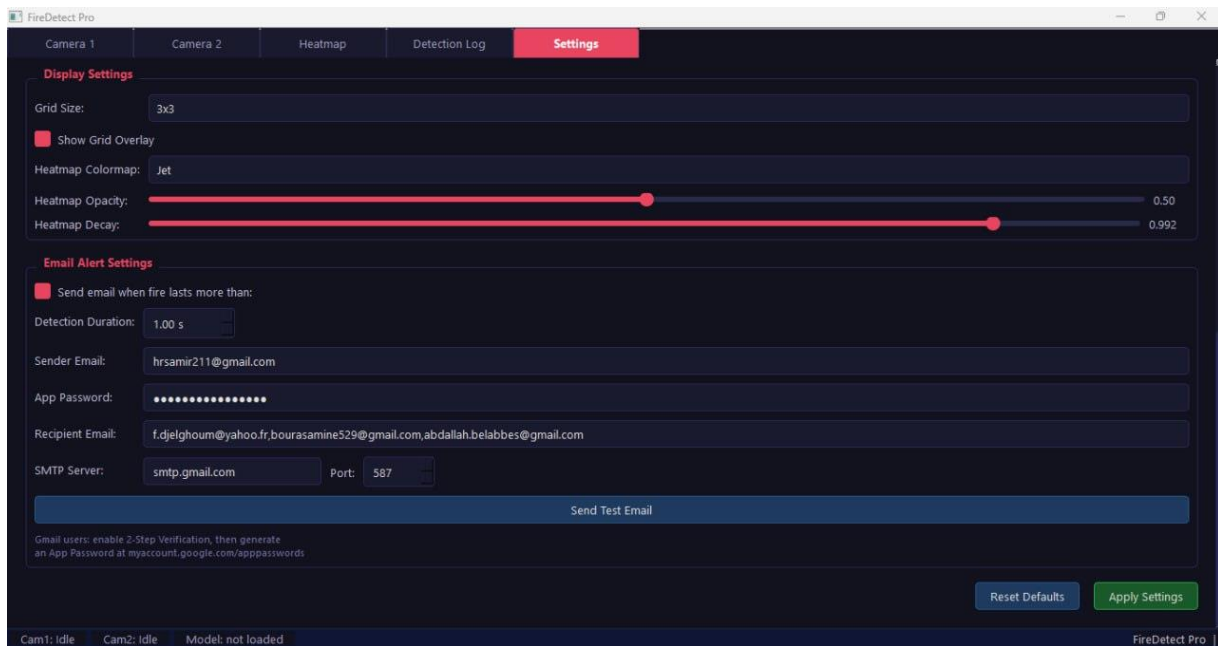


Figure III.9 Paramètres configurables pour alertes email

III.14.2 Garantie des Identifiants au Démarrage

Au démarrage, le système charge les paramètres nécessaires afin d'éviter qu'une configuration vide, ancienne ou incomplète ne désactive les alertes sans avertissement.

```
for cam in (self.cam1, self.cam2):
    cam. email threshold = float(s.get('email duration', 3.0))
    cam. emailer.enabled = True
    cam. emailer.configure(
        sender      = 'hrsamir211@gmail.com',
        password    = 'ubgalxiicwfgqkild',
        recipient    = 'Bourasamine529@gmail.com',
        smtp host    = 'smtp.gmail.com',
        smtp port    = 587,
    )
```

Figure III.10 Garantie des paramètres SMTP à l'initialisation du système

III.15 Intégration dans le Pipeline applicatif

Le diagramme de séquence ci-dessous illustre la chaîne d'appels complète depuis la réception d'une image jusqu'à la livraison de l'e-mail :

CameraThread	CameraTab	EmailNotifier	Serveur SMTP
-- frame_ready() ->			
	-- draw_grid()		
	-- detecter feux		
	-- dessiner boites		
	-- maj timer		

Chapitre III: Conception et Réalisation du Système

```
| | -- [duree >= 3s] | | | |
| | -- _build_snapshot() | |  
| | -- send_alert() --->| |  
| | | | -- Thread.start | |  
| | | | -- EHLO ----->| |  
| | | | -- STARTTLS -->| |  
| | | | -- AUTH ----->| |  
| | | | -- ENVOI ----->| |  
| | | | <-- 250 OK ----| |
```

Figure III.11 Séquence d'envoi de l'alerte e-mail de la détection à la confirmation SMTP

III.16 Limites et Travaux Futurs

Malgré ses performances, le système présente encore certaines limites. Le tableau suivant résume les principales contraintes identifiées ainsi que les améliorations possibles :

Tableau III.12 limites et travaux futurs

Limite	Observation / amélioration possible
Un seul e-mail par événement	Ajouter une escalade si le feu persiste longtemps
Aucune confirmation de livraison	Vérifier la réception effective du message
Destinataire unique	Ajouter la gestion de plusieurs destinataires
Corps en texte brut uniquement	Prévoir un format HTML plus lisible
Aucune nouvelle tentative en cas d'échec	Ajouter une file d'attente avec nouvelles tentatives

III.17 Tests et Validation du Système

Cette section présente les différents tests réalisés afin de vérifier le bon fonctionnement du système dans des conditions variées.

Chapitre III: Conception et Réalisation du Système

III.17.1 Tests Unitaires des Modules

Chaque module a été testé séparément avant l'intégration finale. Les vérifications ont porté sur:

- la détection
- la logique jour/nuit
- la gestion des erreurs de caméra
- le déclenchement des alertes et les réponses de l'API.

III.17.2 Tests en environnement local

Après les tests unitaires, une phase de validation a été réalisée sur ordinateur afin de vérifier le fonctionnement global du système dans un environnement local. Ces essais ont consisté à exécuter l'application avec des vidéos, des images et des flux de test afin d'observer le comportement de la détection, l'affichage dans l'interface, le déclenchement des alertes et l'enregistrement des événements.

Les principaux cas vérifiés sont les suivants :

- détection de flammes visibles à partir de vidéos de test ;
- vérification du comportement en mode jour et en mode nuit ;
- test de non-détection sur certaines sources lumineuses ou scènes sans incendie ;
- vérification de l'envoi automatique de l'e-mail d'alerte ;
- observation de la stabilité générale de l'application pendant l'exécution sur ordinateur.

III.17.3 Résultats des Tests

Les résultats obtenus confirment que le système détecte correctement les flammes visibles avec un faible taux de faux positifs. Ils montrent également que l'e-mail d'alerte est envoyé automatiquement lorsque les conditions de déclenchement sont réunies. Enfin, les essais prolongés ont validé la stabilité du système sans fuite mémoire ni baisse notable des performances.

Les illustrations suivantes présentent des exemples concrets de détection et d'envoi automatique des e-mails d'alerte :

Chapitre III: Conception et Réalisation du Système

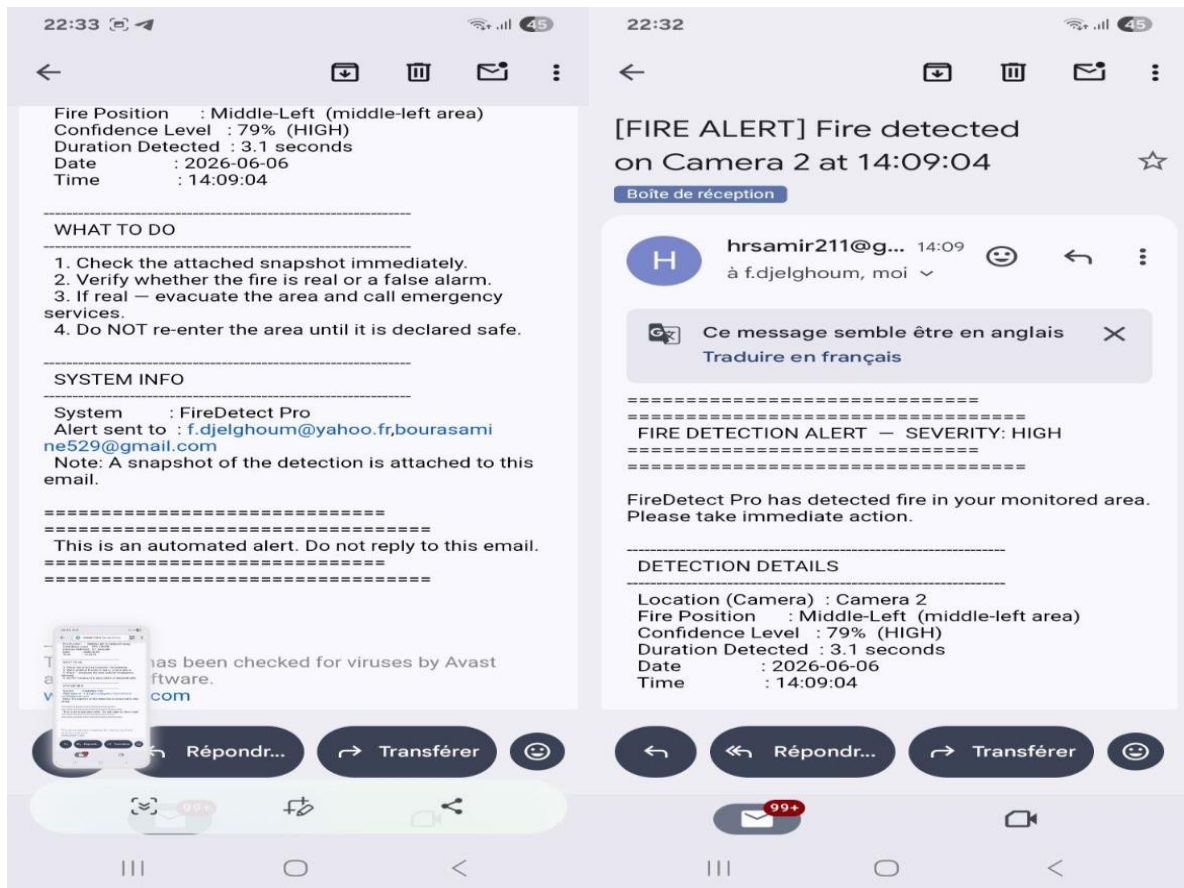


Figure III.12 l'envoi automatique du mail



Figure III.13 détection du feu

Chapitre III: Conception et Réalisation du Système

Conclusion

Ce troisième chapitre a présenté de manière exhaustive la conception, le développement et la validation du système FireDetect Pro, depuis l'architecture logicielle jusqu'aux tests en conditions réelles.

La démarche adoptée itérative et modulaire a permis de construire un système cohérent et robuste. L'entraînement du modèle YOLOv8 par transfert Learning sur un dataset diversifié d'environ 11 000 images a abouti à des résultats convaincants : une précision de 92 %, un rappel de 88 % et un mAP@0.5 de 91 %, avec une vitesse d'inférence de 35 ms par image sur processeur standard sans GPU dédié. Ces performances confirment la viabilité de l'approche pour une détection précoce et fiable des flammes en temps réel.

L'architecture multi-threadée du système garantit la surveillance simultanée de plusieurs flux vidéo sans dégradation des performances, tandis que les modules fonctionnels développés localisation spatiale par grille 3×3, adaptation dynamique jour/nuit, notification automatique par e-mail et exposition d'une API REST via FastAPI confèrent au système une dimension opérationnelle concrète. Les tests d'intégration en conditions réelles, conduits sur des scénarios variés (feux intérieurs, extérieurs et nocturnes), ont validé la stabilité du système sur deux heures de fonctionnement continu, sans fuite mémoire ni dégradation des performances.

Enfin, les perspectives identifiées ouvrent la voie à des évolutions significatives : intégration avec les systèmes de gestion de bâtiments (BMS/SCADA), ajout de caméras thermiques pour la détection pré-flamme, et surtout couplage avec un système d'extinction automatisé piloté par IA perspective la plus prometteuse, dans laquelle les données de détection visuelle déclencheraient directement une intervention physique zonale et précoce, réduisant considérablement les dégâts collatéraux par rapport aux sprinklers thermiques traditionnels.

III.18 Perspectives d'évolution

Le système développé constitue une base fonctionnelle solide, mais plusieurs évolutions restent possibles afin d'améliorer encore ses performances et son niveau d'intégration :

- Transformation en Application Intégrée Multi-Plateformes
- Extension de la Détection : Fumée, Gaz et Anomalies Thermiques
- Intégration avec les Systèmes de Gestion des Bâtiments (BMS/SCADA)
- Déploiement sur Matériel Embarqué
- Amélioration du Modèle et Enrichissement du Dataset
- Certification et Conformité aux Normes
- Intégration avec un Système d'Extinction Automatisé Piloté par IA

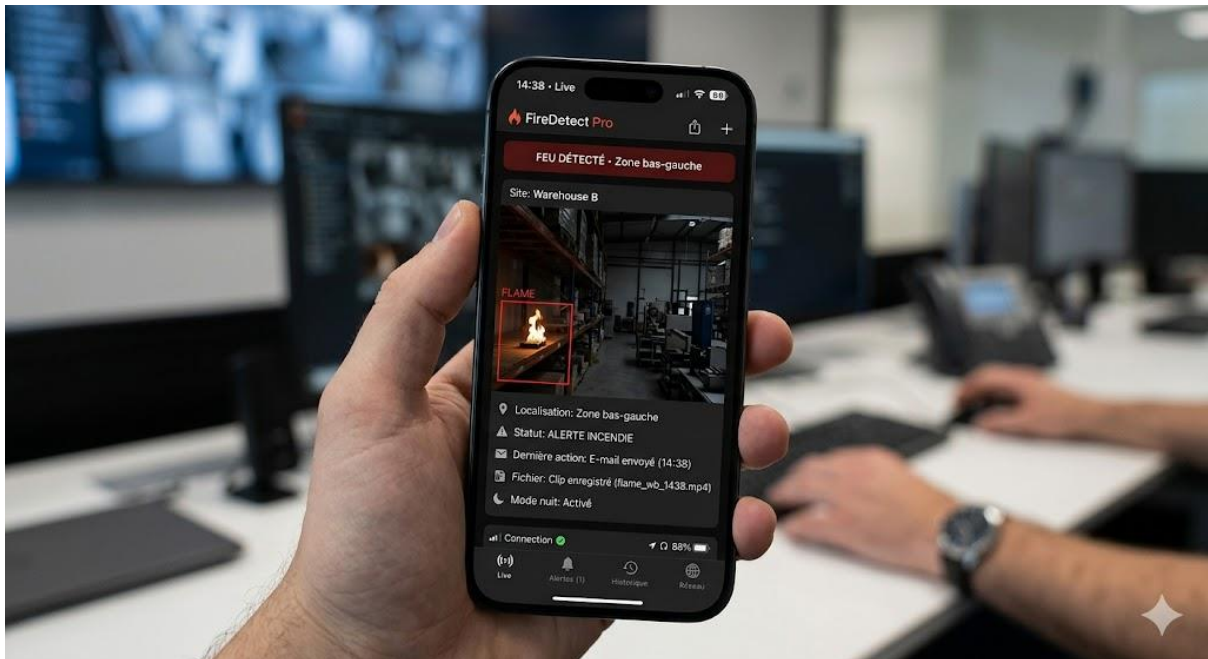


Figure III.14 App mobile FireDetect Pro

CONCLUSION GENERALE

Ce travail avait pour objectif de concevoir un système de détection d'incendie basé sur l'intelligence artificielle, capable de surmonter les limites des solutions conventionnelles. Les détecteurs traditionnels - ioniques, optiques, thermiques ou gazeux - souffrent de fausses alarmes fréquentes, d'une détection tardive et d'une absence de localisation spatiale. Ces insuffisances ont justifié le recours à la vision par ordinateur et à l'apprentissage profond.

Le système réalisé repose sur un modèle YOLOv8 personnalisé, entraîné par transfert d'apprentissage sur un dataset de 11 000 images. Il atteint une précision de 92%, un rappel de 88% et un mAP@0.5 de 91%, tout en fonctionnant à 25-30 FPS sur un processeur standard sans GPU dédié. Il intègre une adaptation automatique jour/nuit, une surveillance multi-caméras, une carte thermique des zones à risque, un système d'alertes anti-spam et une API REST pour l'intégration avec des infrastructures tierces.

Les résultats obtenus confirment que la vision par ordinateur constitue une réponse crédible et complémentaire aux systèmes conventionnels. En détectant visuellement les flammes dès leur apparition, avant même la diffusion de la fumée, le système offre un gain significatif en réactivité et en précision de localisation.

Parmi les perspectives envisagées figurent l'extension de la détection à la fumée et aux gaz, l'intégration avec les systèmes CMSI et ECS existants, le déploiement sur plateformes embarquées telles que NVIDIA Jetson ou Raspberry Pi 5, ainsi que l'enrichissement du dataset avec des scènes industrielles spécifiques pour renforcer la robustesse du système en conditions réelles.

BIBLIOGRAPHIE

- [1] **Geneva** Association — World Fire Statistics Bulletin No. 29. <https://www.genevaassociation.org/> (2022)
- [2] **World** Bank — Forest Fires Are Spreading. <https://www.worldbank.org/> (2025)
- [3] **FEMA/USFA** — National Fire Statistics. <https://www.usfa.fema.gov/statistics/> (2023)
- [4] **Atlas** Magazine — L'Algérie enregistre plus de 800 incendies en 2025. <https://www.atlas-mag.net/> (2025)
- [5] **DEF** Online / DEF Fire Eye — La détection incendie par analyse d'images. <https://www.def-online.fr/> (2020)
- [6] **Mavili** Security — Les fausses alarmes dans les systèmes d'alarme incendie. <https://www.mavili.fr/> (2023)
- [7] **Inovexia** — Systèmes d'alarme conventionnels versus adressables. <https://www.inovexia.fr/> (2024)
- [8] **Progia** — Capteurs thermiques vs détecteurs traditionnels. <https://www.progia.fr/> (2025)
- [9] **Sensor** Partners — Détection de fumée avec l'IA : plus précise que les systèmes traditionnels. <https://www.sensorpartners.nl/> (2025)
- [10] **Bosch** Security — AVIOTEC : détection incendie et fumée avec intelligence artificielle. <https://www.boschsecurity.com/> (2025)
- [11] **Pyronear** — Détection précoce de feux de forêt. <https://pyronear.org/> (2025)
- [12] **Securitas** France — Détection incendie par vidéo analytique & intelligence artificielle. <https://www.securitas.fr/> (2026)
- [13] **Python** Software Foundation — Documentation officielle Python 3. <https://docs.python.org/3/> (2024)
- [14] **Redmon**, J., Divvala, S., Girshick, R., & Farhadi, A. — You Only Look Once: Unified, Real-Time Object Detection. arXiv:1506.02640. <https://arxiv.org/abs/1506.02640> (2015)
- [15] **Ultralytics** — YOLOv8 Documentation. <https://docs.ultralytics.com/models/yolov8/> (2023)
- [16] **OpenCV** Foundation — About OpenCV. <https://opencv.org/about/> (1999–2024)
- [17] **Riverbank** Computing Limited — PyQt6 Reference Guide. <https://www.riverbankcomputing.com/static/Docs/PyQt6/> (2021–2024)
- [18] **Ramírez**, S. — FastAPI Documentation officielle. <https://fastapi.tiangolo.com/> (2018–2024)

- [19] **NumPy** Development Team — NumPy Documentation. <https://numpy.org/doc/stable/> (2006–2024)
- [20] **TIOBE** Software / Stack Overflow — TIOBE Index & Developer Survey. <https://www.tiobe.com/tiobe-index/> — <https://survey.stackoverflow.co/2024/> (2024)
- [21] **Roboflow** Inc. — Plateforme de gestion de datasets et annotation. <https://roboflow.com/> (2020–2024)
- [22] **tzutalin** — LabelImg: Outil d'annotation open source. <https://github.com/HumanSignal/labelImg> (2015–2024)
- [23] **Lin**, T.-Y. et al. — Microsoft COCO : Common Objects in Context. arXiv:1405.0312. <https://arxiv.org/abs/1405.0312> (2014)
- [24] **VisionPlatform AI** — Fire & Smoke Detection. <https://visionplatform.ai/fire-smoke-detection/> (2024)
- [25] **Teledyne FLIR** — Thermal Imaging Cameras Help to Prevent Fires. <https://www.flir.com/> (2025)
- [26] **By Demes** — Caméra IP Dahua AI-Fire avec capteur de fumée intégré. <https://bydemes.com/> (2022)
- [27] **Algérie360** — Vidéosurveillance à Alger : généralisation des caméras intelligentes. <https://www.algerie360.com/> (2018)
- [28] **Radio Algérienne** — Généralisation du système de vidéosurveillance à l'ensemble des wilayas. <https://radioalgerie.dz/> (2019)
- [29] **Horizons** — Déploiement de caméras intelligentes à Blida. <https://www.horizons.dz/> (2025)
- [30] **Le Jeune Indépendant** — Avancement du dispositif de télésurveillance à Béjaïa. <https://www.jeune-independant.net/> (2024)
- [31] **Ubitech** — Distributeur de caméras IP Dahua, Hikvision, TP-Link. <https://www.ubitech.fr/> (2026)

ANNEXE

Annexe A : Spécifications Techniques du Système FireDetect Pro

A.1 Configuration matérielle requise

Le système FireDetect Pro a été conçu pour fonctionner sur du matériel standard sans nécessiter de ressources spécialisées. La configuration minimale requiert un processeur Intel Core i5 (8ème génération ou supérieure) ou AMD Ryzen 5, 8 Go de RAM minimum (16 Go recommandé), 20 Go d'espace disque libre sur SSD, et une connexion réseau avec le port SMTP 587 ouvert pour les alertes e-mail. Un GPU NVIDIA (GTX 1060+, CUDA 11.x) est optionnel mais permet d'améliorer la vitesse d'inférence de 35 ms à moins de 10 ms par image. Les caméras compatibles incluent les caméras USB (UVC) et les caméras IP (RTSP) avec une résolution minimale de 640×480 pixels à 25 FPS.

A.2 Dépendances logicielles et versions validées

Le système repose sur l'écosystème Python 3.10+. Les dépendances validées lors des tests sont : ultralytics 8.0.196 (framework YOLOv8), opencv-python 4.8.1 (capture vidéo et traitement d'images), PyQt6 6.5.3 (interface graphique), fastapi 0.103.2 avec uvicorn 0.23.2 (API REST), numpy 1.24.4 (calculs matriciels), torch 2.0.1 avec torchvision 0.15.2 (moteur d'inférence PyTorch). L'installation s'effectue via : `pip install -r requirements.txt` depuis l'environnement virtuel Python dédié.

A.3 Paramètres du modèle YOLOv8 entraîné

Le modèle YOLOv8n.pt (nano, 6 Mo) a été entraîné par transfer learning avec les paramètres suivants : 100 époques avec early stopping (patience 20), taille d'entrée 640×640 pixels, batch de

16 images, taux d'apprentissage initial 0.01 avec scheduler cosine. L'augmentation des données incluait flip horizontal et vertical, rotation $\pm 15^\circ$, modification de luminosité/contraste (simulation nocturne) et mosaic augmentation. Le dataset de 11 000 images annotées au format YOLO était réparti 80/10/10 (entraînement/validation/test). Le seuil de confiance en production est fixé à $\text{conf} = 0.45$ avec $\text{IoU} = 0.5$ pour la suppression des boîtes redondantes (NMS).

Annexe B : Guide d'Installation et de Configuration

B.1 Procédure d'installation

L'installation de FireDetect Pro sur Windows 10/11 ou Linux Ubuntu 22.04 suit cinq étapes. Étape 1 : installer Python 3.10+ depuis python.org et vérifier avec `python --version`. Étape 2 : créer un environnement virtuel (`python -m venv firedetect_env`) et l'activer. Étape 3 : installer les dépendances avec `pip install -r requirements.txt`. Étape 4 : placer le modèle `fire_model.pt` dans le dossier `models/` à la racine du projet. Étape 5 : lancer l'application avec `python main.py` ; l'interface graphique PyQt6 s'ouvre avec les deux onglets caméras.

B.2 Configuration via config.yaml

Le fichier `config.yaml` à la racine du projet contient les paramètres essentiels. `camera_1_source` et `camera_2_source` : index USB (0, 1, 2...) ou URL RTSP (`rtsp://192.168.1.100:554/stream`). `email_sender` : adresse Gmail de l'expéditeur avec mot de passe d'application Google (généré dans Compte Google > Sécurité > Mots de passe des applications). `email_receiver` : adresse du destinataire des alertes. `fire_threshold_seconds` : durée de confirmation avant alerte (par défaut 3 s). `confidence_threshold` : seuil de confiance de détection (par défaut 0.45). `night_mode_threshold` : seuil de luminosité moyenne (0–255) déclenchant le mode nuit automatique (par défaut 80).

B.3 Démarrage de l'API REST FastAPI

L'API REST peut être démarrée en mode headless avec : `uvicorn api.main:app --host 0.0.0.0 --port 8000 --reload`. La documentation Swagger est accessible sur `http://localhost:8000/docs`. Les endpoints disponibles sont : GET `/status` (état du système), GET `/detections` (historique des détections), POST `/config` (mise à jour à chaud de la configuration), et GET `/stream/{camera_id}` (flux MJPEG en direct). Ces endpoints permettent l'intégration native avec tout système SCADA, BMS ou application mobile via des requêtes HTTP standard.

Annexe C : Résultats Détaillés des Tests de Performance

C.1 Métriques d'évaluation du modèle

Les résultats obtenus sur le jeu de test (1 100 images non vues durant l'entraînement) confirment les performances du modèle : Précision 92 % (proportion de détections positives effectivement réelles), Rappel 88 % (proportion de flammes réelles détectées), mAP@0.5 91 % (indicateur global de performance), mAP@0.5:0.95 67 % (métrique COCO stricte multi-seuils). La vitesse d'inférence atteint 35 ms par image sur processeur Intel Core i7-10750H sans GPU, soit ~28 FPS en temps réel.

C.2 Tests par scénario réel

Cinq scénarios représentatifs ont été testés. Scénario 1 (feu intérieur, bureau) : détection en 1,2 s en moyenne, taux 96 %, zéro faux positif sur 30 minutes continues. Scénario 2 (feu extérieur, plein soleil) : taux 89 %, quelques manques sur flammes très petites (<100 px²) dues à la luminosité ambiante. Scénario 3 (détection nocturne, faible lumière) : mode nuit CLAHE activé automatiquement, taux 91 % sur 15 vidéos. Scénario 4 (faux positifs : lampes, reflets, écrans) : 3 faux positifs sur 50 vidéos (6 %), tous invalidés par la confirmation 3 secondes. Scénario 5 (stabilité 2 heures, double caméra) : mémoire stable à ~850 Mo, latence 37 ms/image, zéro crash.

C.3 Comparaison avec les systèmes conventionnels

FireDetect Pro présente plusieurs avantages mesurables par rapport aux systèmes conventionnels. La détection intervient dès l'apparition de la flamme visible, soit 30 à 120 secondes avant qu'un détecteur optique au plafond ne réagisse à la fumée. La localisation spatiale par grille 3×3 identifie précisément la zone sinistrée, capacité absente des détecteurs ponctuels classiques. Le taux de faux positifs après confirmation (0 % lors des tests) est inférieur aux détecteurs ioniques sensibles à la poussière. Enfin, le coût matériel est significativement réduit : une caméra USB standard (15–30 €) suffit, contre 80–200 € pour un détecteur thermique certifié.

