RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieure et de la Recherche Scientifique
Université Saad Dahlab de Blida 1
Faculté des Sciences
Département de l'Informatique

MASTER EN INFORMATIQUE

Option : Ingénierie des Systèmes Intelligent, et Systèmes Informatique et Réseaux

Présenté par :

Mohamed Mahmoud Ikram

Boumdhal Ahlem

Thème

Intelligence Artificielle pour la Génération du Code Web

Devant le Jury :

Mme. BOUSTIA.N	USDB	Promotrice
Mme. ZAHRA.F	USDB	Co-promotrice
Mme. BACHA	USDB	Présidente
Mme. TOBJI	USDB	Examinatrice

Remerciements

Avant toute chose ,El hamdoullah, qui nous a donné la force, la patience et le courage nécessaires pour mener à bien ce travail.

Nous tenons à exprimer notre profonde gratitude à nos promotrices, Mme Zahra et Mme Boustia, pour leur bienveillance, leur patience, leur écoute et leur présence rassurante tout au long de ce parcours. Leur accompagnement, tant sur le plan scientifique que moral, a été d'un soutien précieux, notamment dans les moments d'hésitation ou de doute.

Enfin, nous remercions également les membres du jury pour avoir accepté d'évaluer notre travail et pour l'attention qu'ils lui ont portée.

Dédicace

Je tiens à exprimer ma gratitude la plus profonde à toute ma famille, en particulier à la famille Boumdhal et à la famille Nahnah, à mes parents, pour leur amour inébranlable et leur confiance silencieuse, qui m'ont portée plus que je ne saurais l'exprimer.

À mes amis, précieux compagnons de route, merci pour vos encouragements, vos mots rassurants. Merci d'avoir tendu l'oreille lorsque mes doutes devenaient trop lourds, et d'avoir su me rappeler que je n'étais pas seule.

Enfin... Merci à moi-même. Pour avoir tenu bon, pour avoir persévéré malgré les doutes, la fatigue, les nuits blanches, pour ne pas m'être abandonnée, même lorsque ma santé m'a trompée. Merci pour chaque petit pas que j'ai su faire, même tremblant. Aujourd'hui, je me rends hommage, avec humilité, tendresse et fierté.

Ahlem

Dédicace

je remercie Dieu pour la force et la patience qu'Il m'a données tout au long de ce parcours, ainsi que moi-même, pour avoir persévéré malgré les doutes, pour chaque nuit blanche, chaque ligne écrite, chaque effort silencieux qui m'a menée jusqu'ici.

Un merci à Ahlem ma binôme, pour son soutien constant, son sérieux et cette complicité sans laquelle ce projet n'aurait pas eu la même saveur.

Je suis profondément reconnaissante envers ma famille mes parents, pour leur amour inconditionnel et leur présence rassurante, et mes frères et sœurs, pour leurs encouragements, leurs sourires, et leur manière unique de me pousser à donner le meilleur de moi-même.

Enfin, une pensée toute spéciale pour Ichrak, ma meilleure amie, pour sa lumière, son écoute, et ses mots qui tombent toujours au bon moment.

Merci à toutes celles et ceux qui, d'une manière ou d'une autre, ont apporté leur pierre à l'édifice de cette aventure.

Ikram

Résumé

La création d'un site web demeure une tâche ardue pour la majorité des utilisateurs. Elle requiert des compétences techniques en développement, en design, en hébergement, ainsi qu'une compréhension des besoins fonctionnels et esthétiques. Ce processus peut être long, coûteux et peu flexible, surtout pour les petites structures qui ne disposent ni des moyens financiers ni des ressources humaines nécessaires. En conséquence, de nombreuses idées ou projets restent bloqués à cause de ces barrières techniques.
Face à cette réalité, ce travail propose une alternative tirer parti du potentiel croissant de l'intelligence artificielle générative pour automatiser la création de sites web. En particulier, notre travail constitue une première étape dans le processus de génération de pages web interactives, dans la perspective de créer ultérieurement des sites web fonctionnels. Cette étape pose les bases nécessaires à l'automatisation de la conception web

Mots Clés : Intelligence Artificielle Générative, Développement , Automatisation de la conception web

Abstract

Creating a website remains a challenging task for most users. It requires technical skills in development, design, and hosting, as well as an understanding of both functional and aesthetic needs. This process can be lengthy, costly, and inflexible especially for small organizations that lack the financial means or human resources. As a result, many ideas or projects remain stalled due to these technical barriers.

In response to this reality, this work proposes an alternative by leveraging the growing potential of generative artificial intelligence to automate website creation. Specifically, our work represents a first step in the process of generating interactive web pages, with the goal of eventually creating fully functional websites. This step lays the necessary foundation for automating web design.

Keywords: Artificial Intelligence, Development, automating web design

ملخص

لا يزال إنشاء موقع إلكتروني مهمة شاقة لغالبية المستخدمين. فهو يتطلب مهارات تقنية في التطوير، التصميم، والاستضافة، بالإضافة إلى فهم للاحتياجات الوظيفية والجمالية. قد تكون هذه العملية طويلة ومكلفة وغير مرنة، خاصةً بالنسبة للهياكل الصغيرة التي لا تملك الموارد المالية أو البشرية اللازمة. ونتيجة لذلك، تبقى العديد من الأفكار أو المشاريع عالقة بسبب هذه الحواجز التقنية.

في مواجهة هذا الواقع، يقترح هذا العمل بديلاً يستفيد من الإمكانات المتزايدة للذكاء الاصطناعي التوليدي لأتمتة إنشاء المواقع الإلكترونية. على وجه الخصوص، يمثل عملنا خطوة أولى في عملية توليد صفحات ويب تفاعلية، بهدف إنشاء مواقع إلكترونية وظيفية في المستقبل. تضع هذه الخطوة الأسس اللازمة لأتمتة تصميم الويب.

الكلمات المفتاحية: الذكاء الاصطناعي توليدي، تطوير الويب، تصميم الويب.

Liste des abréviations

ANN	Artificial Neural Network (Réseau de Neurones Artificiel)
API	Application Programming Interface (Interface de Programmation d'Application)
BLEU	Bilingual Evaluation Understudy (Métrique d'évaluation textuelle)
CNN	Convolutional Neural Network (Réseau de Neurones Convolutionnel)
CORS	Cross-Origin Resource Sharing (Partage de Ressources Cross-Origin)
CSS	Cascading Style Sheets
DL	Deep Learning (Apprentissage Profond)
DNN	Deep Neural Network (Réseau de Neurones Profond)
FIM	Fill-in-the-Middle (Remplissage au Milieu)
GAN	Generative Adversarial Network (Réseau Antagoniste Génératif)
GPU	Graphics Processing Unit (Unité de Traitement Graphique)
HTML	HyperText Markup Language
IA	Intelligence Artificielle
JavaScript	JavaScript (langage de programmation)
JSON	JavaScript Object Notation
LLMs	Large Language Models (Grands Modèles de Langage)
LoRA	Low-Rank Adaptation
LSTM	Long Short-Term Memory (Mémoire à Long Terme)
NLP	Natural Language Processing (Traitement du Langage Naturel)
Q&A	Question and Answer (Question et Réponse)
RBF	Radial Basis Function (Fonction de Base Radiale)
RNN	Recurrent Neural Network (Réseau de Neurones Récurrent)
ROUGE	Recall-Oriented Understudy for Gisting Evaluation (Métrique d'évaluation textuelle)

SEO	Search Engine Optimization (Optimisation pour les Moteurs de Recherche)
SNN	Spiking Neural Network (Réseau de Neurones Spiking)
SOTA	State Of The Art (État de l'Art)
UI	User Interface (Interface Utilisateur)
VAE	Variational Autoencoder (Autoencodeur Variationnel)
VRAM	Video Random Access Memory (Mémoire Vidéo à Accès Aléatoire)

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Motivation et problématique

Le développement logiciel, bien qu'essentiel à l'ère numérique, reste un domaine complexe et souvent inaccessible à une large partie de la population. En effet, coder nécessite une expertise technique approfondie, une maîtrise de divers langages de programmation, ainsi qu'une compréhension des concepts informatiques avancés. Pour les non-initiés, cela représente un véritable obstacle. De plus, le coût lié à l'apprentissage ou à l'embauche de développeurs qualifiés est souvent élevé, ce qui freine considérablement l'innovation et l'automatisation, surtout dans les environnements à ressources limitées.

Cette problématique devient encore plus critique lorsqu'il s'agit du développement web. Aujourd'hui, plus de 70% des entreprises possèdent un site web, et selon les statistiques, il existe plus de 1,13 milliard de sites web dans le monde, contre environ 5 millions d'applications mobiles. Ces chiffres témoignent de l'importance majeure du web comme principal canal de communication, de marketing et de service. Pourtant, pour les petites entreprises, artisans, commerçants et freelances, créer un site web moderne, esthétique et fonctionnel reste un défi. Le manque de compétences techniques, le coût des services professionnels, et la complexité des outils actuels limitent leur accès à cette vitrine numérique pourtant cruciale.

Objectif

L'objectif global est de rendre la création de sites web aussi simple et accessible que possible, tant pour les particuliers que pour les entreprises, en automatisant la génération du code à base de l'intelligence artificielle. En supprimant les barrières techniques liées au développement traditionnel, nous permettons à des utilisateurs de concevoir facilement un site web fonctionnel, simplement en exprimant leurs besoins en langage naturel. Notre projet se concentre principalement sur la génération de pages web, en s'appuyant sur les

langages fondamentaux du développement front-end tels que HTML, CSS et JavaScript. Il s'agit d'une phase initiale essentielle qui ouvre la voie à la création automatique de sites web complets et pleinement fonctionnels.

Organisation du mémoire

Chapitre 1

Ce chapitre introduit les concepts fondamentaux de l'intelligence artificielle générative, en mettant l'accent sur son application dans la génération de code. Il explore les bases de l'apprentissage automatique, les réseaux de neurones et les modèles de langage , en détaillant leur rôle dans l'automatisation du développement web.

Chapitre 2

Ce chapitre détaille les concepts techniques sous-jacents au projet, notamment l'ingénierie des prompts pour interagir avec les modèles, l'architecture des Transformers, ainsi que le choix du modèle utilisé et l'exploration de son architecture.

Chapitre 3

Ce chapitre décrit la méthodologie adoptée, incluant la structure de travail, la collecte des jeux de données, et le processus d'entraînement du modèle, en présentant les différentes étapes de préparation, de configuration et d'ajustement du modèle.

Chapitre 4

Ce chapitre présente l'évaluation du modèle à travers différentes méthodes (automatique, A/B, humaine), l'interface utilisateur développée pour faciliter l'interaction avec le système, ainsi qu'une comparaison avec les systèmes existants.

Chapitre 1

IA Générative pour la génération du code

1.1 Introduction

L'intelligence artificielle occupe aujourd'hui une place centrale dans l'innovation technologique. Ce chapitre vise à fournir une compréhension des principes de base de l'IA générative. Nous mettrons particulièrement l'accent sur l'IA générative pour la génération de code. Ce sujet constitue le fil conducteur de cette étude.

1.2 Apprentissage automatique

L'apprentissage automatique relève le défi de concevoir des ordinateurs capables de s'améliorer de manière autonome grâce à l'expérience [1]. Dans ce domaine, des algorithmes et des modèles sont développés pour permettre aux ordinateurs d'apprendre et d'extraire des connaissances à partir des données [2]. Cette capacité d'adaptation repose sur différents types d'apprentissage, chacun correspondant à une manière particulière d'interagir avec les données et l'environnement [3], tels que :

- Apprentissage supervisé
- Apprentissage non supervisé
- Apprentissage semi-supervisé
- Apprentissage par renforcement
- Apprentissage par transfert

1.2.1 Apprentissage profond

L'apprentissage profond (deep learning) est une branche importante de l'apprentissage automatique (machine learning) qui vise à automatiser l'apprentissage des caractéristiques et la reconnaissance des motifs en construisant et en entraînant des modèles de réseaux de neurones à plusieurs couches [2, 4].

1.3 Réseau de neurones

Un réseau de neurones (RN) est un système d'opérateurs non linéaires interconnectés, recevant des signaux de l'extérieur par ses entrées, et délivrant des signaux de sortie. Ces réseaux de neurones sont une métaphore des structures cérébrales, de traitement parallèle, de distribution d'information, et comportent plusieurs éléments de traitement appelés neurones [5].

1.3.1 Architecture de réseau de neurones

Un réseau neuronal se compose de plusieurs neurones ordonnés dans des plans appelés couches [6]. Un réseau de neurones se compose de couches de nœuds (neurones). Chaque nœud reçoit une entrée, la traite, puis la transmet à la couche suivante (voir Figure 1.1) [7].

— **Couche d'entrée (Input Layer)** : La première couche qui reçoit les données brutes (par exemple, des images, du texte).
— **Couches cachées (Hidden Layers)** : Les couches situées entre l'entrée et la sortie, qui transforment les données et détectent des motifs.
— **Couche de sortie (Output Layer)** : Produit la prédiction finale.

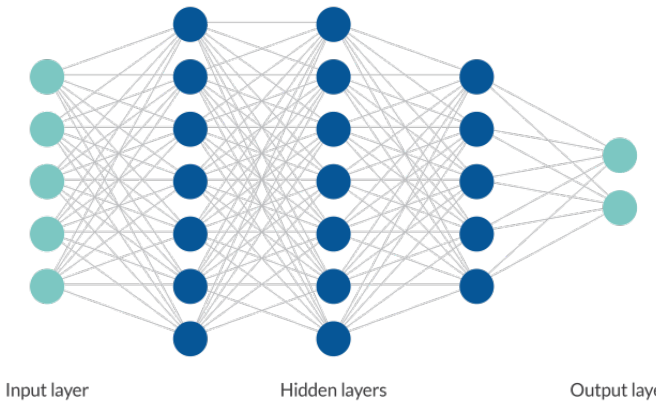

Figure 1.1 – Architecture d'un réseau de neurones [7]

1.3.2 Différents types de réseaux de neurones

Type de Réseau de Neurones	**Caractéristique Clé**	**Cas d'Utilisation Courants**
Réseau Feedforward	Flux de données unidirectionnel	Prédictions simples, tâches de base
Réseau Convolutionnel (CNN)	Détecte des motifs visuels	Reconnaissance d'images, imagerie médicale
Réseau Récurrent (RNN)	Gère les données séquentielles	Séries temporelles, reconnaissance vocale
Mémoire à Long Terme (LSTM)	Se souvient des dépendances à long terme	Génération de texte, chatbots
Unité Récurrente Gated (GRU)	LSTM simplifié	Traitement du langage naturel (NLP), prédiction de séries temporelles
Fonction de Base Radiale (RBF)	Utilise la fonction de base radiale	Classification, reconnaissance de motifs
Réseau Antagoniste Génératif (GAN)	Génère de nouvelles données	Génération d'images, deep fakes
Transformeur	Mécanisme d'auto-attention	Modèles linguistiques, traduction
Réseau Modulaire	Sous-réseaux indépendants multiples	Processus complexes multi-tâches
Réseau de Neurones Spiking (SNN)	Traitement basé sur le temps	Robotique, calcul neuromorphique

Tableau 1.1 – Types de Réseaux de Neurones et leurs Caractéristiques

1.4 IA Générative

L'intelligence artificielle générative désigne des systèmes d'intelligence artificielle capables de créer du texte, des images ou d'autres formes de médias grâce à l'utilisation de modèles génératifs. Ces modèles acquièrent une compréhension des motifs et des structures présents dans leurs données d'entraînement, puis génèrent de nouvelles données présentant des caractéristiques similaires [8]. L'intelligence artificielle générative englobe différents types de modèles, chacun étant conçu pour des tâches spécifiques ou pour la

génération de formes particulières de médias [8]. Dans le domaine du développement logiciel, l'IA générative exploite des technologies telles que le traitement du langage naturel (NLP) et les grands modèles de langage (LLMs) pour aider les développeurs à générer du code, compléter des fonctions ou écrire des programmes entiers à partir de descriptions. Elle permet également la traduction de code entre langages, les tests automatisés, la complétion de code et la refactorisation de bases de code existantes [9].

1.4.1 Architecture des Modèles Génératifs

Les modèles génératifs peuvent être classés selon différentes familles architecturales, chacune répondant à des objectifs spécifiques et présentant des avantages distincts. Les modèles autorégressifs, par exemple, génèrent des séquences en prédisant chaque élément conditionnellement aux éléments précédents ; des architectures telles que *GPT*, *PaLM* ou *LLaMA* relèvent de cette catégorie.

Les modèles de diffusion, quant à eux, apprennent à inverser un processus de dégradation progressive afin de produire du contenu. [10]Initialement conçus pour la génération d'images, comme *DALL·E 2* ou *Stable Diffusion*, ils peuvent néanmoins être adaptés à des tâches textuelles.

Les réseaux génératifs adverses (*GANs*) reposent sur un entraînement antagoniste entre un générateur et un discriminateur ; s'ils sont principalement appliqués à l'image, certaines variantes comme *SeqGAN* ou *GSGAN* ont été développées pour le texte.

Les autoencodeurs variationnels (*VAEs*), de leur côté, combinent encodage et décodage autour d'un espace latent structuré de manière probabiliste.

De nouvelles approches émergent également, telles que les modèles à états (*State Space Models*), dont *Mamba* constitue un exemple prometteur pour le traitement efficace de longues séquences. [11]

Enfin, les *Transformers* représentent aujourd'hui la base de la plupart des modèles génératifs modernes centrés sur le texte. [12, 13]

1.4.2 Méthodologies des Modèles Génératifs

Cette section présente les principales approches méthodologiques employées dans la mise en œuvre des modèles génératifs :

Génération Autoregressive

La génération autoregressive constitue la technique principale utilisée dans les LLMs modernes. Le processus génère une séquence token par token, où chaque nouveau token

est prédit en fonction de tous les tokens précédents dans la séquence. [14]

Décodage

Plusieurs stratégies de décodage sont employées pour contrôler la génération :

— **Greedy Decoding** : Sélection du token le plus probable à chaque étape.
— **Beam Search** : Maintien de plusieurs hypothèses parallèles.
— **Top-k Sampling** : Échantillonnage parmi les k tokens les plus probables.
— **Nucleus Sampling (Top-p)** : Échantillonnage dans un sous-ensemble dynamique basé sur la probabilité cumulée. [14]

Optimisation

— **Gradient Clipping** : Prévention de l'explosion des gradients.
— **Learning Rate Scheduling** : Ajustement adaptatif du taux d'apprentissage.
— **Mixed Precision Training** : Utilisation de précisions numériques mixtes pour optimiser la mémoire. [15]

1.4.3 Modèles de langage

Les modèles de langage représentent une avancée significative dans le domaine de l'intelligence artificielle, capables de comprendre et de générer du texte semblable à celui des humains, grâce à un entraînement sur des données massives [16]. Ces modèles sont formés sur de vastes ensembles de données couvrant divers sujets, langues et styles, ce qui leur permet d'accomplir une large gamme de tâches liées au langage, telles que la traduction, la synthèse, la création de contenu et même la réponse à des questions complexes. Leur polyvalence en fait des outils précieux dans différents secteurs, et récemment, leur application dans les secteurs non lucratifs et humanitaires a attiré l'attention en raison de leur potentiel à avoir un impact positif significatif [17, 18].

1.4.4 Modèles de langage pour la génération du code

Les modèles de langage de grande taille, grâce à leurs puissantes capacités de raisonnement, ont été largement adoptés dans la recherche et les applications liées au code [19]. Les LLMs basés sur l'architecture Transformeur ont révolutionné de nombreux domaines, et leur application à la génération de code a eu un impact particulièrement significatif.

Ces modèles suivent un processus complet, qui commence par la curation et la synthèse des données de code, suivi d'une approche d'entraînement structurée incluant le pré-

entraînement, l'ajustement fin, l'apprentissage par renforcement avec différents types de rétroaction, ainsi que l'utilisation de techniques avancées de conception de prompts [20].

Les avancées récentes incluent l'intégration de la génération de code augmentée par récupération d'informations (retrieval-augmented code generation) à l'échelle des dépôts de code (repository-level), ainsi que le développement d'agents autonomes de codage.

Par ailleurs, l'évaluation des capacités de codage des LLMs est devenue un élément essentiel de ce domaine de recherche [20].

1.4.4.1 Exemples d'Outils d'IA Générative pour la Génération du Code

Plusieurs outils puissants de génération de code assistée par intelligence artificielle ont gagné en popularité pour leur capacité à aider les développeurs à coder plus efficacement. Voici quelques exemples :

- **IBM watsonx Code Assistant :** Cet outil aide les développeurs à générer du code grâce à des recommandations générées par l'IA, adaptées à leurs besoins spécifiques, quel que soit leur niveau d'expérience.
- **GitHub Copilot :** Outil de complétion de code alimenté par l'IA, intégré à des environnements de développement populaires. Il propose des extraits de code contextuels dans plusieurs langages de programmation, améliorant ainsi la productivité et réduisant les erreurs.
- **Qwen Coder :** Modèle de génération de code multilingue développé par Alibaba, conçu pour comprendre des requêtes complexes et produire du code structuré dans plusieurs langages.
- **DeepSeek Coder :** Modèle open-source performant axé sur la génération de code à partir d'instructions naturelles, avec des capacités avancées de compréhension de contexte.
- **StarCoder :** Développé dans le cadre du projet BigCode, StarCoder est un modèle LLM spécialisé dans la génération de code et la compréhension de bases de code volumineuses, tout en respectant les principes d'IA responsable.
- **WPCode :** Spécialement conçu pour les développeurs WordPress, WPCode génère des extraits de code personnalisés afin d'étendre les fonctionnalités et de faciliter le développement.
- **Codeium :** Assistant de codage alimenté par l'IA qui propose des complétions et des suggestions dans divers langages de programmation, optimisant l'expérience de développement.
- **Deep TabNine :** Version avancée de TabNine exploitant l'apprentissage profond pour générer des lignes de code complètes selon le contexte et les entrées de l'uti-

lisateur.

1.4.4.2 Applications de l'IA Générative dans le Développement Web

L'IA générative a émergé comme une technologie transformative, capable de produire du contenu original à partir de données d'entraînement massives [21]. Dans le domaine du développement web, l'IA générative automatise des tâches traditionnellement manuelles, telles que la conception de structures HTML, la création de styles CSS, et l'implémentation de fonctionnalités JavaScript. En s'appuyant sur des grands modèles de langage (LLMs), ces outils permettent de générer des sites web fonctionnels et esthétiques à partir de prompts en langage naturel. Cette section explore les applications spécifiques de l'IA générative dans le développement web, en détaillant les processus, les outils, les opportunités, les défis, et les implications sociétales [21, 22].

1.4.4.3 Avantages de l'automatisation par l'IA

- **Rapidité :** L'IA peut produire un site web en quelques secondes, contre plusieurs jours ou semaines pour une création manuelle.
- **Accessibilité :** Aucune compétence en codage n'est requise, permettant aux utilisateurs non techniques de créer leur site.
- **Rentabilité :** Des offres abordables réduisent le besoin de faire appel à des développeurs.
- **Fonctionnalités :** Les outils d'IA proposent l'optimisation SEO, des designs responsifs et la création de contenu, améliorant l'expérience utilisateur [23].

1.4.4.4 Limites

Les sites générés par l'IA peuvent manquer de la flexibilité des sites codés sur mesure, limitant les fonctionnalités avancées ou les conceptions uniques. Les utilisateurs recherchant des fonctionnalités complexes peuvent devoir combiner les outils d'IA avec des ajustements manuels [23].

1.5 Revue des recherches sur la génération du code

Afin de mieux situer notre étude dans le contexte actuel de la recherche, nous présentons ci-dessous un ensemble de travaux récents qui explorent l'utilisation des modèles de langage de grande taille (LLMs) dans la génération automatique de code, en particulier dans le développement web. Ces recherches permettent d'identifier les capacités, les

avantages ainsi que les limitations de ces technologies émergentes.

— **Compréhension du HTML avec des modèles de langage de grande taille :** L'article examine comment les modèles de langage de grande taille (LLMs) peuvent être utilisés pour comprendre le HTML à travers trois tâches : la classification sémantique des éléments HTML, la génération de descriptions textuelles, et la navigation autonome sur des pages HTML. Les résultats montrent une nette amélioration par rapport aux approches classiques, notamment avec moins de données. Cependant, l'article reconnaît certaines limitations. Premièrement, il souligne la dépendance aux données d'entraînement, car l'efficacité des LLMs dépend fortement de la qualité et de la quantité des données utilisées pour le fine-tuning. Deuxièmement, la complexité des pages HTML peut poser des défis, car les modèles peuvent avoir des difficultés avec des pages très complexes ou mal structurées, ce qui pourrait affecter leur capacité à générer des descriptions précises. Enfin, bien que les LLMs soient efficaces, ils ne surpassent pas toujours les modèles spécialisés qui ont été formés avec des méthodes d'apprentissage par renforcement, nécessitant souvent plus d'interactions en ligne pour atteindre des performances optimales. [24]

— **Génération automatisée du code frontend avec OpenAI :** Cet article explore l'utilisation de l'API d'OpenAI pour la génération automatique de code front-end (HTML, CSS, JavaScript), visant à améliorer l'efficacité du développement web. La méthode permet de réduire le temps de création des interfaces utilisateur, notamment pour des prototypes ou des applications simples. Toutefois, les auteurs soulignent que cette technologie nécessite une supervision humaine pour garantir la qualité, la conformité aux exigences, et la gestion de projets complexes. Les principales limites identifiées sont la précision dépendante des prompts, les difficultés avec des interfaces plus sophistiquées, et le besoin d'une intervention humaine pour assurer la cohérence et la fiabilité du code généré. Ces solutions automatisées offrent un potentiel prometteur, mais ne remplacent pas encore complètement l'expertise humaine dans le développement web. [25]

— **Expérimentations de génération du code web avec des LLMs : Une étude comparative :** Le modèle utilisé, Gemini 1.5 Pro, est un grand modèle de langage développé par Google DeepMind, basé sur l'architecture Transformer, conçu pour la compréhension multimodale et la génération de code web. Il a été entraîné sur de vastes datasets, notamment du code et du texte technique, pour produire des résultats précis et accessibles. Cependant, ses limites incluent des difficultés à gérer des exigences complexes en accessibilité, surtout dans des tâches interactives ou contextuelles, et la nécessité d'un prompt optimisé pour obtenir une meilleure précision. [26]

— **DeepSeek Coder : Quand le grand modèle de langage rencontre la programmation :** DeepSeek-Coder est une série de modèles de langage open-source conçus pour améliorer l'intelligence de code dans le développement logiciel. Ces modèles, allant de 1,3 milliard à 33 milliards de paramètres, ont été entraînés sur un corpus de code de haute qualité contenant 2 trillions de tokens. Ils utilisent une tâche de remplissage de blanc avec une fenêtre de 16K, ce qui améliore leur capacité à générer et compléter du code. Les évaluations montrent que DeepSeek-Coder surpasse même des modèles fermés comme Codex et GPT-3.5, et il est disponible sous une licence permissive pour une utilisation commerciale et de recherche. Cependant, DeepSeek-Coder présente des limitations. La performance des modèles dépend de la qualité des données d'entraînement, et ils peuvent avoir des difficultés avec des tâches de programmation complexes. De plus, l'utilisation de ces modèles nécessite des ressources computationnelles importantes, ce qui peut restreindre leur accessibilité pour certains utilisateurs. [27]

— **L'IA générative dans le développement web front-end : Une comparaison entre l'IA comme outil et l'IA comme programmeur autonome :** L'article présente ChatGPT comme un outil puissant pour la génération de texte, le codage et l'assistance au développement logiciel, notamment en web et programmation. Cependant, ses performances sont variables, avec des risques de code non fonctionnel ou sous-optimisé, et il peut parfois mal interpréter des contextes complexes. Les limites principales incluent la fiabilité des réponses, la qualité inconstante, et les biais présents dans les données d'entraînement. L'étude recommande donc une utilisation complémentaire à l'expertise humaine, en tenant compte de ces limitations. [28]

Acticle	**Approche**	**Modèle**	**Tâches principales**
Compréhension du HTML avec des modèles de langage de grande taille	Affinage de T5 pour classification HTML, génération de descriptions et navigation web	T5	Classification sémantique HTML Génération de descriptions Navigation autonome

Article	**Approche**	**Modèle**	**Tâches principales**
Génération automatisée du code frontend avec OpenAI	Utilisation de l'API OpenAI pour générer du HTML/CSS/JS depuis le langage naturel	OpenAI GPT-3/4	Génération d'UI Prototypage rapide Création e-commerce
Expérimentations de génération du code web avec des LLMs	Utilisation de LLMs pour générer du code web et améliorer l'accessibilité WCAG	Gemini 1.5 Pro (Google DeepMind)	Génération HTML/CSS/JS Amélioration accessibilité WCAG Correction automatique Apprentissage de la programmation Évaluation de modèles LLM
DeepSeek Coder : Quand le grand modèle de langage rencontre la programmation	Développement d'une série de modèles open-source entraînés sur 2T tokens avec tâche de remplissage de blanc	DeepSeek-Coder (1,3B à 33B)	Génération de code source Remplissage de blanc dans du code existant Résolution de problèmes complexes Évaluation de benchmarks Développement open-source
L'IA générative dans le développement web front-end	Étude sur 3 pages web créées avec aide humaine vs 100% IA	ChatGPT-4 (OpenAI)	Génération complète de pages web Assistance à la programmation Évaluation (temps, code, UX)

Tableau 1.3 – Comparaison des approches et modèles pour la génération de code

Parmi les travaux analysés, **DeepSeek-Coder** apparaît comme le modèle le plus performant pour la génération de code complexe, bien qu'il nécessite des ressources importantes. **Gemini 1.5 Pro** se démarque par sa précision et sa gestion de l'accessibilité, mais reste dépendant de bons prompts. **ChatGPT** est utile en tant qu'outil d'assistance, mais présente une qualité variable. L'**API d'OpenAI** est efficace pour des prototypes simples, mais nécessite supervision. Enfin, les **LLMs** de l'article sur *HTML* montrent un bon potentiel de compréhension, mais restent limités pour la génération directe de code web.

1.6 Conclusion

Ce chapitre présente les bases essentielles de l'IA générative, qui permet de produire automatiquement du contenu. Ces notions posent les fondations nécessaires à la compréhension du projet. Dans le prochain chapitre, nous verrons comment générer du code à partir d'un prompt utilisateur, à l'aide de modèles transformers.

Chapitre 2

Technologies Clés de l'IA Générative

2.1 Introduction

Dans ce chapitre, nous explorons les fondements de l'intelligence artificielle générative, en mettant en avant l'importance de l'ingénierie des prompts. Une attention particulière est portée à l'architecture des transformers, dont le fonctionnement est essentiel au traitement des données séquentielles. Cette étude nous a également permis de motiver le choix du modèle utilisé dans notre projet.

2.2 Ingénierie des Prompts

L'ingénierie des prompts est l'art de concevoir, formuler et affiner les instructions données aux modèles d'IA générative afin de les guider pour produire des résultats utiles, précis et pertinents. Un prompt est simplement une instruction ou une question en langage naturel – cela peut aller d'un simple mot à un scénario complexe – qui indique à l'IA ce que vous attendez d'elle [29].

2.2.1 L'efficacité de l'ingénierie des prompts

— Les prompts sont clairs, spécifiques et riches en contexte. L'IA est plus susceptible de générer des réponses correctes et pertinentes si le prompt ne laisse pas place à la confusion.

— Des flux de travail en plusieurs étapes sont nécessaires. Des techniques avancées comme le « chain-of-thought prompting » (chaîne de raisonnement) peuvent guider l'IA à travers une logique étape par étape, améliorant ainsi la précision pour des tâches mathématiques ou logiques.

— Les biais et les erreurs doivent être minimisés. L'ingénierie des prompts peut réduire le risque de réponses biaisées, hors sujet ou inventées en formulant les requêtes de manière à orienter l'IA loin des contenus indésirables [29].

2.2.2 Les limites de l'ingénierie des prompts

L'ingénierie des prompts échoue lorsque :

— Les prompts sont flous, ambigus ou manquent de contexte. L'IA générera des réponses hors sujet, génériques ou dénuées de sens si elle ne comprend pas bien la demande.

— Les prompts sont trop complexes ou alambiqués. Des prompts trop longs ou compliqués peuvent désorienter le modèle ou nuire à la tâche ciblée, produisant ainsi des résultats moins précis.

— L'utilisateur s'appuie uniquement sur la modification du prompt pour corriger des problèmes systémiques. Certains problèmes – comme les biais présents dans les données d'entraînement ou les limites du modèle – ne peuvent pas être corrigés uniquement en ajustant le prompt [29].

2.3 Les architectures de l'IA générative

Trois grandes architectures sous-tendent l'IA générative :

— **Réseaux antagonistes génératifs (GANs)** : Utilisent deux réseaux de neurones (un générateur et un discriminateur) qui s'affrontent, menant à la création de données synthétiques réalistes.

— **Autoencodeurs variationnels (VAEs)** : Codent les données d'entrée en une représentation compressée, puis les décodent, apprenant ainsi à générer de nouvelles données similaires aux originales.

— **Transformers** : Le progrès le plus révolutionnaire récent, en particulier pour les tâches impliquant des données séquentielles telles que le langage, les images et l'audio [30]. L'accent sera mis sur les architectures à décodeur seul des Transformers.

2.4 Introduction au modèle Transformer

Les transformers sont un type d'architecture de deep learning qui a révolutionné la manière dont les systèmes d'IA traitent et génèrent des données séquentielles. Contrairement aux anciens modèles (comme les RNN et les LSTM) qui traitaient les séquences étape par étape, les transformers traitent l'ensemble de la séquence en parallèle. Cette parallélisation

est rendue possible grâce au mécanisme d'auto-attention, qui permet au modèle d'évaluer l'importance de chaque élément de la séquence par rapport aux autres [31].
Développés pour surmonter les limites des RNN telles que la lenteur du traitement séquentiel et la difficulté à capturer efficacement les dépendances à long terme, les transformers utilisent l'auto-attention pour accélérer l'apprentissage et mieux gérer les relations complexes dans les données [32].

2.4.1 L'architecture Transformer

Les transformers sont basés sur une architecture **encodeur-décodeur**, chacun composé de plusieurs couches identiques.

- L'**encodeur** traite la séquence d'entrée pour générer des représentations contextuelles.
- Le **décodeur** génère ensuite la séquence de sortie, souvent de manière autorégressive (un token à la fois), comme dans les tâches de génération de texte.

Chaque couche contient plusieurs composants clés, que l'on peut comprendre comme suit :

- **Auto-attention multi-têtes** : C'est le mécanisme central, qui permet au modèle de pondérer l'importance des différents tokens d'entrée en fonction de leurs relations. Il utilise un processus appelé attention, dans lequel le modèle se concentre sur les tokens pertinents — un peu comme un humain qui se focalise sur certains mots pour comprendre une phrase. Par exemple, dans la phrase « Le chat a faim et il veut du lait », le modèle peut se concentrer sur « chat » lorsqu'il traite « il ». Ce processus est effectué en parallèle sur plusieurs « têtes », ce qui permet de capturer différents types de relations.
- **Réseaux feed-forward positionnels** : Ce sont des étapes de traitement supplémentaires qui transforment la sortie de la couche d'attention, permettant au modèle d'affiner sa compréhension de chaque token en tenant compte du contexte.
- **Connexions résiduelles et normalisation de couche** : Ces mécanismes assurent une formation stable en ajoutant l'entrée à la sortie de chaque sous-couche et en normalisant les activations, ce qui évite au modèle de se « bloquer » pendant l'apprentissage.
- **Encodage positionnel** : Étant donné que les transformers traitent les séquences en parallèle (sans ordre implicite), des encodages positionnels sont ajoutés aux embeddings d'entrée pour indiquer la position de chaque mot. C'est comme ajouter des repères pour dire que la séquence est « Le chat a faim » et non « Faim a chat le ». [31]

2.4.2 Flux de traitement

Le fonctionnement des transformers suit plusieurs étapes coordonnées qui permettent au modèle de traiter et de générer des séquences efficacement, comme illustré dans le diagramme ci-dessus.

- **Tokenisation**
  - La séquence d'entrée est d'abord divisée en unités plus petites appelées tokens (mots ou sous-mots), puis chaque token est converti en un vecteur d'embedding.
- **Calcul de l'attention**
  - Chaque token produit des **requêtes (Q)**, **clés (K)** et **valeurs (V)**.
  - Les scores d'attention sont calculés en comparant les requêtes aux clés. Ces scores déterminent l'influence que chaque token a sur les autres pendant l'encodage contextuel.
- **Calcul parallèle**
  - Contrairement aux modèles récurrents qui fonctionnent séquentiellement, les transformers traitent tous les tokens en parallèle, ce qui améliore l'efficacité de l'entraînement et de l'inférence.
- **Représentation contextuelle via l'auto-attention**
  - Les couches d'auto-attention permettent à chaque token de prendre en compte tous les autres tokens de la séquence d'entrée, ce qui aide à capturer les dépendances à longue portée et le contexte global.
- **Interaction encodeur-décodeur**
  - L'encodeur transforme la séquence d'entrée en une représentation riche et contextuelle.
  - Le décodeur utilise cette représentation, ainsi que les tokens générés précédemment, pour prédire le prochain token dans la séquence de sortie.
- **Ajout de l'information de position**
  - Puisque le traitement parallèle élimine la notion d'ordre, des encodages positionnels sont ajoutés aux embeddings pour expliciter la position des mots.
- **Génération de sortie**
  - Le décodeur produit les tokens un par un, en utilisant les mécanismes d'attention pour intégrer à la fois l'entrée et les sorties précédentes (génération autorégressive). [31]

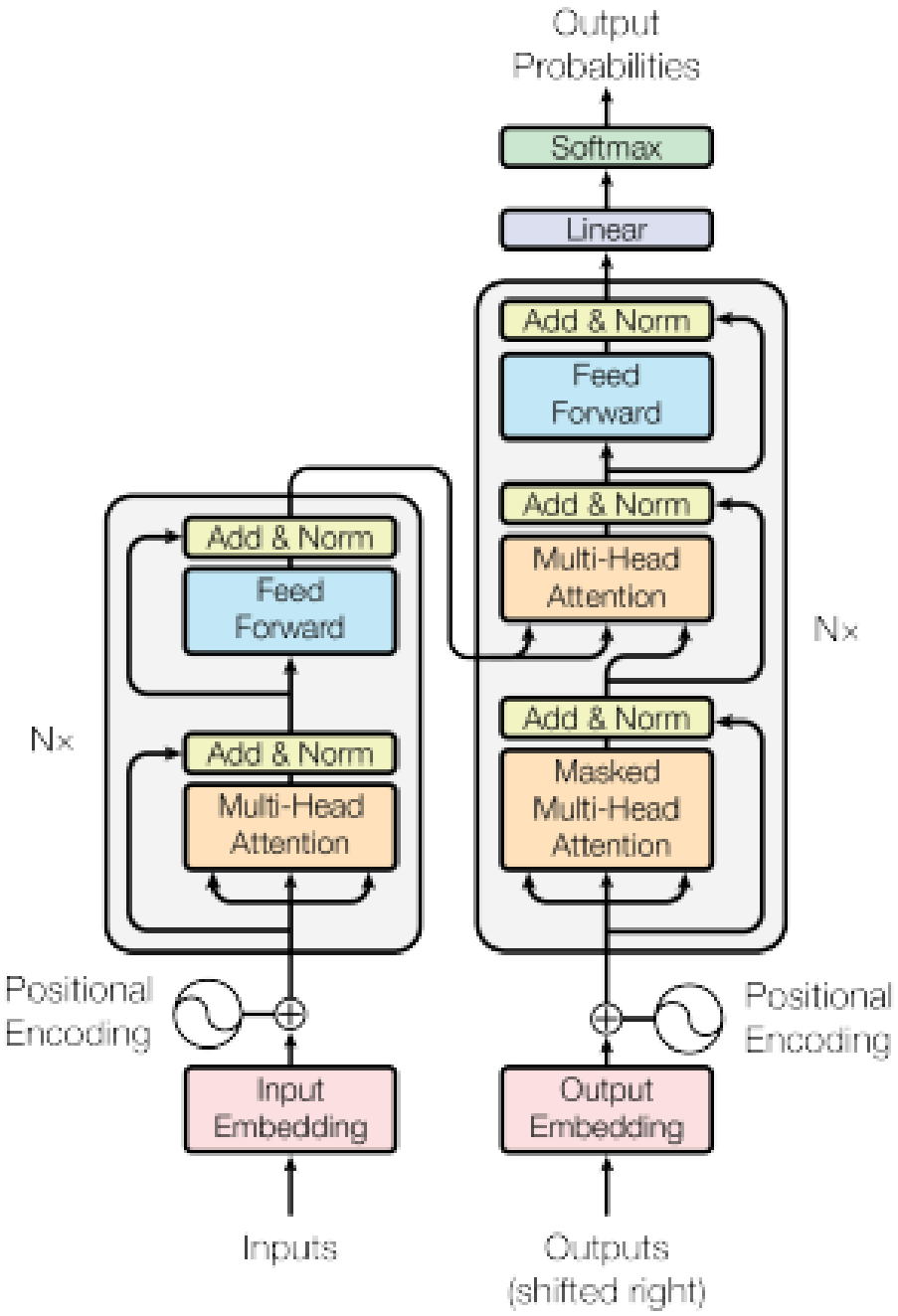

FIGURE 2.1 – Architecture du modèle Transformer [31]

2.4.3 Limites des transformeurs

— **Entrées de longueur fixe** : Nécessitent de découper les longues séquences, ce qui peut entraîner une perte de contexte.

— **Coût computationnel** : Exigent beaucoup de mémoire et d'énergie.

— **Besoins en données** : Nécessitent de grands ensembles de données pour le pré-entraînement.
— **Interprétabilité** : Les mécanismes d'attention complexes sont difficiles à interpréter [33].

2.5 Techniques d'Adaptation des Modèles

Bien que l'architecture Transformer constitue une base solide pour l'IA générative, les modèles pré-entraînés nécessitent souvent une adaptation pour des tâches spécifiques ou des domaines particuliers. Cette adaptation permet d'exploiter les connaissances générales acquises lors du pré-entraînement tout en spécialisant le modèle pour des applications ciblées. Les techniques d'adaptation varient en termes de complexité computationnelle, d'efficacité mémoire et de performance, offrant différents compromis selon les contraintes du projet [34].

2.5.1 Fine-tuning

Le fine-tuning, ou ajustement fin, consiste à continuer l'entraînement d'un modèle pré-entraîné sur un ensemble de données spécifique à la tâche cible. Cette approche tire parti de l'apprentissage par transfert, où les représentations apprises sur de vastes corpus de données générales sont adaptées à des domaines plus spécialisés [35].

2.5.1.1 Mécanisme du Fine-tuning

Le processus de fine-tuning implique plusieurs étapes critiques. Premièrement, un modèle pré-entraîné sur un large corpus (comme GPT ou BERT) sert de point de départ. Ce modèle a déjà appris des représentations linguistiques générales et des patterns complexes. Deuxièmement, les poids du modèle sont ajustés en utilisant un taux d'apprentissage généralement plus faible que lors du pré-entraînement, permettant une adaptation graduelle sans perdre les connaissances préalablement acquises [36].

2.5.1.2 Applications et Domaines

Le fine-tuning s'est révélé particulièrement efficace dans de nombreux domaines. En traitement du langage naturel, il permet d'adapter des modèles généraux à des tâches spécifiques comme l'analyse de sentiment, la reconnaissance d'entités nommées ou la traduction automatique. Dans le domaine médical, le fine-tuning de modèles pré-entraînés sur des

corpus biomédicaux a montré des résultats prometteurs pour l'extraction d'informations cliniques et le diagnostic assisté [37].

2.5.1.3 Limitations du Fine-tuning

Malgré son efficacité, le fine-tuning présente plusieurs limitations. L'oubli catastrophique est un défi majeur : le modèle peut perdre ses connaissances générales en s'adaptant à une tâche spécifique, surtout si le dataset de fine-tuning est très différent des données d'origine [38].
Le coût computationnel est également important : le fine-tuning nécessite de mettre à jour tous les paramètres du modèle, ce qui demande des ressources considérables en mémoire et en temps de calcul, particulièrement pour les modèles de grande taille [39].

2.5.2 Une Approche Efficiente (LORA)

Pour répondre aux limites du fine-tuning traditionnel, LoRA propose une approche plus efficiente pour adapter les grands modèles de langage tout en réduisant les coûts computationnels [40].

2.5.2.1 Principe Mathématique de LoRA

LoRA repose sur l'idée que les changements nécessaires pour l'adaptation peuvent être représentés par des matrices de rang faible. Plutôt que de modifier tous les paramètres du modèle, LoRA ajoute une mise à jour sous la forme de deux matrices de rang inférieur. Cela permet de ne modifier qu'un très petit nombre de paramètres, rendant l'adaptation bien plus légère [40].

2.5.2.2 Implémentation et Avantages

LoRA réduit drastiquement les besoins en mémoire GPU, rendant possible l'adaptation sur du matériel plus accessible [41]. De plus, plusieurs modules LoRA peuvent être entraînés indépendamment pour différentes tâches et facilement interchangés, ce qui facilite le déploiement multi-tâches.
Enfin, malgré sa légèreté, LoRA conserve des performances proches de celles du fine-tuning complet, tout en réduisant le temps d'entraînement et la taille des fichiers à stocker [42].

2.5.3 Comparaison des Méthodes d'Adaptation

2.5.3.1 Efficacité Computationnelle

LoRA surpasse largement le fine-tuning en termes d'efficacité. Pour un modèle de 7 milliards de paramètres, LoRA avec un rang de 16 n'utilise que quelques millions de paramètres supplémentaires — une réduction de plus de 99% des paramètres entraînables [40]. Cela se traduit par des gains significatifs en temps d'entraînement et en consommation de mémoire.

2.5.3.2 Performance et Qualité

Le fine-tuning complet reste légèrement supérieur en termes de performance brute, notamment sur des tâches très complexes. Cependant, LoRA atteint souvent 95 à 99% de ces performances avec seulement 0.1% des paramètres entraînables, ce qui en fait un excellent compromis [43].

2.5.3.3 Recommandations d'Usage

Le fine-tuning est à privilégier lorsque la performance maximale est recherchée et que les ressources ne sont pas limitées. En revanche, LoRA est idéal pour des applications pratiques, le prototypage rapide ou des contextes à ressources limitées.
Pour les usages expérimentaux ou très contraints, d'autres techniques plus légères comme le prompt-tuning ou les adapters peuvent être considérées, bien qu'elles soient généralement moins performantes [44].

2.6 Choix du modèle de base

Les modèles génératifs les plus connus comme **GPT (OpenAI)** [45,46], **Gemini (Google DeepMind)**, **Claude (Anthropic)** et même **BERT** [47] (à travers ses variantes adaptées à la génération) s'appuient tous sur l'architecture *Transformer*, majoritairement en mode *decodeur seul* (GPT, Claude, Gemini) ou *encodeur-décodeur* (Gemini pour le multimodal). Ces modèles ont montré une grande efficacité dans la génération de code web (HTML, CSS, JavaScript), notamment grâce à leur capacité à comprendre le contexte, proposer des solutions interactives et générer des interfaces respectant les bonnes pratiques d'accessibilité.
Cependant, malgré leurs performances, ils ne sont pas *open-source*, et leur accessibilité est limitée (utilisation restreinte via API, coût élevé, non *fine-tunable* localement), ce qui en fait de mauvais candidats comme base d'un modèle personnalisable.

En parallèle, des modèles *open-source* comme **LLaMA**, **Qwen-Coder**, **DeepSeek-Coder** et **StarCoder** reposent également sur des architectures de type *Transformer*, avec des variantes optimisées pour le code. Ces modèles sont accessibles, modifiables et entraînables localement, ce qui en fait des candidats adaptés pour servir de modèle de base.

Nous avons donc expérimenté et comparé ces modèles afin de sélectionner le plus adapté à notre cas d'usage. Le tableau suivant présente davantage de détails techniques sur leurs architectures et caractéristiques.

Modèle	**Type d'Architecture**	**Capacité de génération de code web**	**Fine-tuning possible**	**Remarque technique**
Qwen2.5-Coder	Décodeur seulement	Moyen	oui (open source)	équilibre performance/taille, open source
Code Llama	Décodeur seulement	Moyen	oui	Performances correctes, mais modèles plus anciens. Moins optimisé pour les styles, accès parfois limité selon les versions.
StarCoder	Décodeur seulement	Moyen à bon	Oui	Accès limité à StarCoder2, support instable, nécessite une configuration complexe.
DeepSeek-Coder	Décodeur seulement	Bon mais généraliste	Oui mais lourd	Modèle très performant, surtout en génération CSS, mais très exigeant en ressources matérielles. Peu adapté à un environnement à budget limité. Trop lourd pour notre infrastructure.

Tableau 2.1 – Comparaison des modèles de génération de code

Nous avons finalement choisi le modèle Qwen2.5-Coder 0.5B Instruct pour plusieurs raisons essentielles :

— **Compétences Avancées en Programmation** Qwen2.5-Coder, une évolution de CodeQwen1.5, a été entraîné sur plus de 5,5 trillions de tokens, démontrant des performances exceptionnelles en génération de code, raisonnement et correction de code. Il surpasse systématiquement des modèles plus volumineux sur plus de dix benchmarks [48].

- **Opportunité d'Amélioration sur l'Entraînement HTML/CSS** Malgré ses points forts, les données d'entraînement de Qwen2.5-Coder comprennent peu de code HTML et CSS, ce qui conduit à des performances sous-optimales pour générer des sites web complets et interactifs. Cette lacune offre une opportunité de peaufiner le modèle avec des ensembles de données spécialisés afin d'améliorer ses capacités en développement web [49].
- **Flexibilité : Open-Source** Publié sous licence Apache 2.0, Qwen2.5-Coder permet un fine-tuning sans restriction et un déploiement commercial, ce qui en fait un choix idéal pour développer des applications spécialisées en développement web [48].
- **Taille légère (0.5B)** : idéale pour l'entraînement et l'inférence sur des machines à ressources limitées.
- **Opportunités spécifiques pour les étudiants** : Alibaba Cloud met à disposition des étudiants et des chercheurs des ressources éducatives et des opportunités pour explorer et utiliser les modèles Qwen, notamment :
  - **Accès gratuit aux modèles Qwen2.5-Coder** : Les étudiants peuvent accéder gratuitement aux différentes versions de Qwen2.5-Coder via des plateformes telles que Hugging Face et ModelScope, facilitant ainsi l'expérimentation et l'apprentissage.
  - **Documentation et tutoriels détaillés** : Une documentation complète, accompagnée de tutoriels et d'exemples pratiques, est disponible pour aider les étudiants à comprendre et à utiliser efficacement le modèle dans divers scénarios d'application.

2.7 Architecture du Modèle Qwen2.5 Coder

L'architecture de *Qwen2.5-Coder* est directement dérivée de celle de *Qwen2.5*. Le tableau présente l'architecture de *Qwen2.5-Coder* pour six tailles de modèle différentes : 0.5B, 1.5B, 3B, 7B, 14B et 32B paramètres. Bien que toutes ces tailles partagent la même architecture en termes de taille de tête, elles diffèrent sur plusieurs autres aspects clés. À quelques exceptions près, comme le modèle 1.5B qui dispose d'une taille intermédiaire plus grande, ou le modèle 3B qui possède davantage de couches, la majorité des paramètres augmentent généralement à mesure que la taille du modèle croît.
Par exemple, en comparant les modèles 7B et 32B : le modèle 7B présente une taille cachée de 3584, tandis que celle du modèle 32B atteint 5120. Le modèle 7B utilise 28 têtes de requêtes et 4 têtes clé-valeur, tandis que le modèle 32B utilise 40 têtes de requêtes et

8 têtes clé-valeur, ce qui reflète une capacité renforcée. De même, la taille intermédiaire évolue avec la taille du modèle, atteignant 18944 pour le modèle 7B et 27648 pour le modèle 32B.
Par ailleurs, les modèles plus petits utilisent un couplage d'embedding (*embedding tying*), contrairement aux modèles plus grands. Tous les modèles disposent d'un vocabulaire de 151646 tokens et ont été entraînés sur un corpus de 5,5 trillions de tokens [19].

Configuration	0.5B	1.5B	3B	7B	14B	32B
Hidden Size	896	1,536	2048	3,584	5120	5120
# Layers	24	28	36	28	48	64
# Query Heads	14	12	16	28	40	40
# KV Heads	2	2	2	4	8	8
Head Size	128	128	128	128	128	128
Intermediate Size	4,864	8,960	4,864	18,944	13824	27648
Embedding Tying	✓	✓	✓	✗	✗	✗
Vocabulary Size	151,646	151,646	151,646	151,646	151,646	151,646
# Trained Tokens	5.5T	5.5T	5.5T	5.5T	5.5T	5.5T

Tableau 2.2 – Tableau d'architecture de Qwen2.5-Coder [50]

Tokenisation

Qwen2.5-Coder hérite du vocabulaire de *Qwen2.5*, mais introduit plusieurs tokens spéciaux pour aider le modèle à mieux comprendre le code. Le tableau suivant donne un aperçu des tokens spéciaux ajoutés lors de l'entraînement afin de mieux capturer les différentes formes de données de code. Ces tokens remplissent des fonctions spécifiques dans le processus de traitement du code.
Par exemple, `<|endoftext|>` marque la fin d'un texte ou d'une séquence, tandis que les tokens `<|fim_prefix|>`, `<|fim_middle|>` et `<|fim_suffix|>` sont utilisés pour la technique de *Fill-in-the-Middle* (FIM), où le modèle doit prédire les parties manquantes d'un bloc de code. Le token `<|fim_pad|>` est utilisé comme remplissage lors des opérations FIM.
D'autres tokens comme `<|repo_name|>`, qui identifie les noms de dépôts, et `<|file_sep|>`, utilisé comme séparateur de fichiers, permettent de mieux gérer les informations à l'échelle d'un dépôt. Ces tokens sont essentiels pour aider le modèle à apprendre à partir de structures de code variées et lui permettent de traiter des contextes plus longs et plus complexes, que ce soit au niveau des fichiers ou des dépôts [50].

Token	Token ID	Description
<|endoftext|>	151643	end of text/sequence
<|fim_prefix|>	151659	FIM prefix
<|fim_middle|>	151660	FIM middle
<|fim_suffix|>	151661	FIM suffix
<|fim_pad|>	151662	FIM pad
<|repo_name|>	151663	repository name
<|file_sep|>	151664	file separator

Tableau 2.3 – Tableau des tokens spéciaux [50]

2.7.1 Modélisation causale du langage

La modélisation causale du langage, dans *Qwen-Coder*, désigne la génération auto-régressive de code où chaque token est prédit en fonction des tokens précédents, en respectant la structure causale du code source telle que les dépendances entre variables, fonctions et instructions [51].

2.7.2 Données de Préentraînement

Des données à grande échelle, de haute qualité et diversifiées constituent la base des modèles préentraînés. Dans cette optique, un jeu de données nommé *Qwen2.5-Coder-Data* a été construit. Ce jeu de données comprend cinq types de données principaux : données de code source, données texte-code associées, données synthétiques, données mathématiques et données textuelles [19].

2.7.3 Capacités de Qwen2.5-Coder en génération de sites web

Qwen2.5-Coder démontre des capacités avancées dans la génération de code, y compris pour des applications web. Bien qu'il n'existe pas d'études spécifiques détaillant l'utilisation de *Qwen2.5-Coder* pour la création complète de sites web, ses performances sur divers benchmarks de génération de code suggèrent une aptitude à générer des composants web tels que des pages HTML (voir Figure 2.2), des styles CSS et des scripts JavaScript [19].

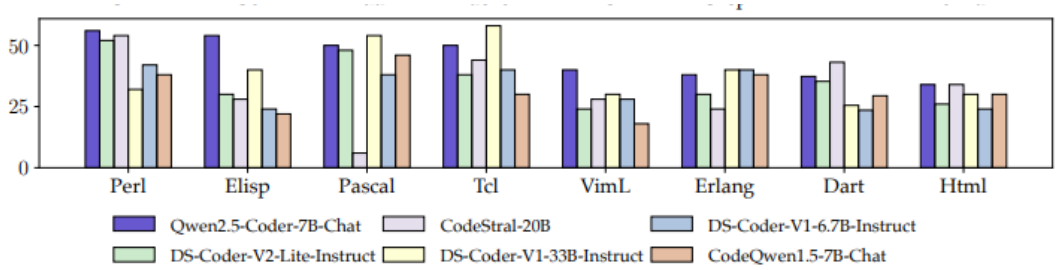

FIGURE 2.2 – Performance Comparative de Modèle Qwen2.5 Coder sur Différents Langages de Programmation [19]

2.7.4 Limitations observées

Malgré l'inclusion de HTML, CSS et JavaScript dans le corpus d'entraînement, des limitations subsistent :

- **Génération de structures simples** : Le modèle tend à produire des structures HTML basiques, souvent similaires d'une génération à l'autre.
- **Styles CSS rudimentaires** : Les styles générés manquent de complexité et de diversité, limitant l'esthétique des sites produits.
- **Manque de personnalisation** : Les sites générés présentent peu de variations en termes de mise en page et de design.

2.7.5 Présentation de Qwen2.5-Coder Instruct

L'équipe Qwen a développé *Qwen2.5-Coder*, un outil qui vous aide à coder davantage et à apprendre plus. *Qwen2.5-Coder* fait partie de la série *Qwen2.5*, disponible en six tailles de modèles : 0.5B, 1.5B, 3B, 7B, 14B et une version 32B [19].
Ils ont publié la dernière version, nommée *Qwen2.5-Coder 32B Instruct*, le 12 novembre 2024 [19].
Ce modèle ne se contente pas seulement d'aider à coder et à apprendre davantage, il se distingue aussi par plusieurs qualités : puissant, diversifié et pratique [51].

2.7.5.1 Qwen2.5-Coder 0.5B Instruct

Qwen2.5-Coder 0.5B Instruct fait partie de la série *Qwen2.5-Coder*, qui comprend six tailles de modèles : 0.5B, 1.5B, 3B, 7B, 14B et 32B.
Ces modèles sont conçus pour assister dans les tâches de programmation et offrent des capacités puissantes, diversifiées et pratiques [52].

	Qwen2.5 Coder – 0.5B Base
HumanEval	28.0
MBPP-3shot	40.4
MultiPL-E (Average)	24.7
CRUXEval-O (CoT)	23.0
BigCodeBench (Complete Avg)	10.4
Fill-in-the-Middle (Avg)	77.7
GSM8K (4-shot)	34.5
MATH (4-shot)	15.4
MMLU	42.0
ARC-Challenge	34.4

	Qwen2.5 Coder – 0.5B Instruct
HumanEval	61.6
MBPP	52.4
EvalPlus (Average)	53.8
MultiPL-E	49.6
McEval	18.7
LiveCodeBench (2024.07–11)	2.0
CRUXEval-O (CoT)	27.8
BigCodeBench (Instruct Average)	34.5
Spider	55.7
BIRD-SQL	15.1
CodeArena (vs. GPT-4 Turbo 0409)	5.5

FIGURE 2.3 – Déffirence pincipale entre le modèle qwen2.5 coder base et instruct [52]

2.7.5.2 Principales caractéristiques de Qwen2.5-Coder 0.5B Instruct

- **Puissant :**
  - Atteint des performances à la pointe de la technologie (SOTA) parmi les modèles open source en génération de code, réparation de code et raisonnement sur le code.
  - Offre des résultats comparables à GPT-4o pour la génération et la réparation de code [53].
  - Prend en charge plus de 40 langages de programmation [19], y compris des langages spécialisés comme Haskell et Racket.
- **Diversifié :**
  - Disponible en plusieurs tailles pour répondre aux besoins variés des développeurs et aux contraintes de ressources.
  - Le modèle 0.5B comprend 0,49 milliard de paramètres, dont 0,36 milliard hors embeddings, 24 couches, 14 têtes d'attention (dont 2 pour les clés-valeurs), et une longueur de contexte de 32 000 tokens [53].
- **Pratique :**
  - Affiche de bonnes performances dans des scénarios concrets, tels que les assistants de codage et la gestion d'artefacts.
  - Convient aux applications en périphérie grâce à sa petite taille et à ses faibles exigences en ressources.

2.7.5.3 Configuration du modèle Qwen2.5-Coder-0.5B-Instruct

Le modèle *Qwen2.5-Coder-0.5B-Instruct*, basé sur l'architecture Qwen2 pour la modélisation causale du langage, est conçu pour la génération efficace de code et le suivi d'instructions. Avec une taille cachée compacte de 896 dimensions et 24 couches de transformateurs, il offre un équilibre entre efficacité computationnelle et capacité à capturer des motifs complexes dans le code et le texte.

Le modèle utilise un mécanisme d'attention multi-têtes avec 14 têtes d'attention et 2 têtes de clés-valeurs, exploitant une attention par groupes pour optimiser l'utilisation de la mémoire. Ses couches de projection avancées s'étendent à une taille intermédiaire de 4864 unités, utilisant la fonction d'activation SiLU pour modéliser efficacement les relations non linéaires.

Une fenêtre de contexte étendue de 32 768 jetons, soutenue par des embeddings de position rotatifs (RoPE) avec un theta de 1 000 000, permet le traitement de longues séquences de code. La normalisation RMS avec un epsilon de 1e-06 garantit la stabilité de l'entraînement, tandis qu'un vocabulaire de 151 936 jetons prend en charge divers langages de programmation et instructions en langage naturel.

L'utilisation du type de données BFloat16 améliore l'efficacité computationnelle, et les embeddings de mots liés réduisent les besoins en mémoire. La mise en cache est activée pour optimiser l'inférence, et le modèle est construit avec la version 4.51.3 de la bibliothèque `Transformers`, assurant une compatibilité avec les cadres modernes.

Cette configuration fait de *Qwen2.5-Coder-0.5B-Instruct* un modèle léger mais puissant pour les tâches de codage, adapté aux environnements à ressources limitées [53, 54].

```
🔷 Model config:
Qwen2Config {
  "architectures": [
    "Qwen2ForCausalLM"
  ],
  "attention_dropout": 0.0,
  "bos_token_id": 151643,
  "eos_token_id": 151645,
  "hidden_act": "silu",
  "hidden_size": 896,
  "initializer_range": 0.02,
  "intermediate_size": 4864,
  "max_position_embeddings": 32768,
  "max_window_layers": 21,
  "model_type": "qwen2",
  "num_attention_heads": 14,
  "num_hidden_layers": 24,
  "num_key_value_heads": 2,
  "rms_norm_eps": 1e-06,
  "rope_scaling": null,
  "rope_theta": 1000000.0,
  "sliding_window": 32768,
  "tie_word_embeddings": true,
  "torch_dtype": "bfloat16",
  "transformers_version": "4.51.3",
  "use_cache": true,
  "use_sliding_window": false,
  "vocab_size": 151936
}
```


Figure 2.4 – Configuration du modèle Qwen2.5 coder 0.5b Instruct

2.7.5.4 Points forts des performances

— **Mathématiques et tâches de codage :**
  — Montre des améliorations significatives par rapport aux versions précédentes, ce qui le rend adapté aux environnements limités en ressources.
  — Surpasse les autres modèles de sa catégorie dans les benchmarks de mathématiques et de programmation.

- **Compréhension du langage :**
  - Fournit des résultats compétitifs dans les tâches de compréhension du langage, malgré un nombre de paramètres inférieur à certains concurrents.

2.7.5.5 Qwen2.5-Coder Instruct pour la création de sites web

Bien que le modèle *Qwen2.5-Coder 0.5B Instruct* ait démontré de solides capacités en génération de code, débogage et programmation multilingue, il présente certaines limites lorsqu'il s'agit de générer des sites web :

- **Capacités de style limitées :** Le modèle ne peut pas créer de sites web avec plusieurs styles ou des designs très sophistiqués. Il peut générer des pages web fonctionnelles, mais les éléments de design peuvent ne pas répondre aux normes esthétiques avancées.
- **Entraînement basé sur des templates :** Le modèle semble avoir été entraîné sur une variété de modèles et d'exemples de sites interactifs, mais cela ne se traduit pas nécessairement par la création de designs personnalisés ou visuellement impressionnants. Il est mieux adapté aux projets simples et structurés par template.
- **Erreurs possibles dans des scénarios complexes :** Lorsqu'il traite des applications web complexes ou très interactives, le modèle peut générer un code contenant des erreurs ou nécessitant d'importants ajustements manuels, en particulier pour des fonctionnalités JavaScript ou CSS complexes.

En résumé : *Qwen2.5-Coder 0.5B Instruct* est un outil performant pour les tâches de codage basiques à intermédiaires, y compris la génération de sites web simples, mais il n'est pas optimisé pour créer des sites avec des styles avancés ou des fonctionnalités très interactives. Pour des besoins plus complexes, les développeurs devront peut-être s'appuyer sur des modèles plus grands ou des outils complémentaires.
C'est justement là que notre travail entre en jeu : nous allons utiliser *Qwen2.5-Coder 0.5B Instruct* et l'adapter afin qu'il soit capable de générer des sites web modernes, avec des styles esthétiques avancés et des fonctions interactives.

2.8 Conclusion

Dans ce chapitre, nous avons posé les bases théoriques nécessaires à la compréhension de l'intelligence artificielle générative, en mettant particulièrement l'accent sur le rôle des prompts et l'architecture des transformers. Grâce à cela, nous sommes désormais en mesure d'aborder l'implémentation pratique de notre projet avec une vision plus claire. Dans le chapitre suivant, nous allons entrer dans la phase concrète de notre travail.

Chapitre 3

Approche Proposée

3.1 Introduction

Dans ce chapitre, nous explorons le processus de création et d'optimisation de notre modèle pour la génération de code. Nous y présentons en détail la préparation des jeux de données (datasets), les différentes étapes d'entraînement du modèle, ainsi que la structure générale du projet.

3.2 Architecture du système

La figure ci-dessous (Figure 4.5) représente la structure générale de notre travail, allant de la collecte des datasets jusqu'à l'obtention du résultat final sous forme d'une page web en `HTML`, `CSS` et `JavaScript`.
D'abord, nous allons détailler et mettre davantage l'accent sur les étapes liées aux datasets, à l'entraînement du modèle ainsi qu'à son évaluation.

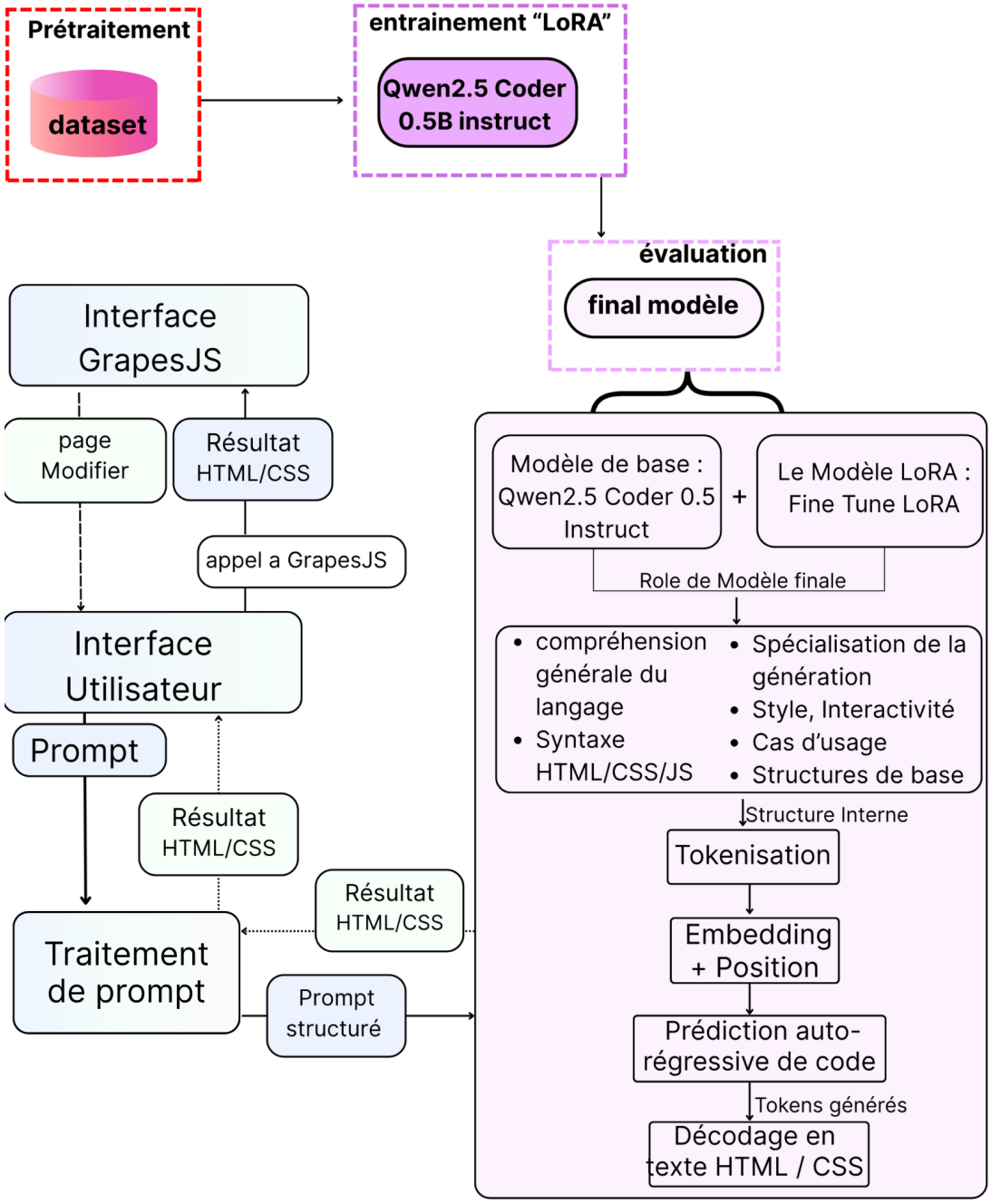

FIGURE 3.1 – Structure globale de projet

Nous allons maintenant expliquer en détail les différentes étapes représentées dans le schéma.

3.3 Collecte des datasets

Pour l'entraînement de notre modèle, nous avons adopté trois méthodes afin de constituer notre jeu de données final. Aucun dataset complet regroupant des templates

HTML/CSS/JavaScript avec leurs prompts n'était disponible.

3.3.1 Dataset Sémantique

3.3.1.1 Attribution de sujets

Objectif : Catégoriser les prompts en sujets sémantiquement significatifs.
Processus :

- Utilisation de Top2Vec pour regrouper les prompts par similarité sémantique.
- En cas d'échec, utilisation de mots-clés (home, portfolio, contact).
- Ajout du sujet attribué à chaque prompt.

3.3.1.2 Construction d'un framework CSS sémantique

Objectif : Créer un framework CSS réutilisable basé sur les sujets.
Processus :

- Regroupement des prompts par sujet.
- Dérivation de propriétés visuelles à partir des mots.
- Génération de classes CSS spécifiques.
- Compilation dans un fichier CSS unifié.

3.3.1.3 Structuration sémantique du dataset HTML

Objectif : Générer des snippets HTML stylisés à partir des prompts.
Processus :

- Utilisation du framework CSS avec les prompts étiquetés.
- Génération de snippets HTML avec titre et contenu.
- Application des classes CSS correspondantes.
- Structuration en label, code HTML, prompt original et sujet.

3.3.1.4 Construction de pages HTML complètes

Objectif : Créer des pages web complètes à partir des snippets HTML.
Processus :

- Assemblage de pages complètes selon le sujet.
- Ajout de sections : en-tête, pied de page, navigation, formulaires.
- Ajout de composants comme grilles de blog, FAQ, bannières.

3.3.1.5 Formatage du dataset de prompts

Objectif : Convertir le dataset HTML en format Q&A.
Processus :

- Conversion de chaque page HTML en prompt-question.
- Le code HTML devient la réponse.

3.3.1.6 Résumé du workflow global

1. **Préparation des entrées** : Dataset de prompts structurés.
2. **Attribution de sujets** : Via Top2Vec ou mots-clés.
3. **CSS framework** : Généré par sujet à partir des mots.
4. **Snippets HTML** : Créés avec les styles thématiques.
5. **Pages HTML complètes** : Composées selon contexte du sujet.
6. **Format Q&A** : Pour entraînement et évaluation.

3.3.1.7 Avantages du dataset

- Apprentissage du format et de la structure d'un site web.
- Traduction de descriptions utilisateur en code.
- Meilleure compréhension du contexte via la notion de sujet.

3.3.1.8 Limitations du dataset

- Style minimal, visuellement peu riche.
- Contenu HTML peu complexe.
- Faible exposition à la diversité CSS.

3.3.2 Dataset Composants et Templates

Afin de pallier les limitations identifiées dans le dataset sémantique, nous avons créé un deuxième dataset plus riche et plus varié, visant à améliorer la qualité et la diversité des exemples fournis.

3.3.2.1 Intégration de Composants Web

Objectif : Constituer une base de composants HTML variés et réalistes.

Processus :

- Sélection d'un dataset Hugging Face contenant des composants web variés.

3.3.2.2 Intégration d'Exemples Riches en JavaScript

Objectif : Enrichir le dataset avec des comportements interactifs JS.

Processus :

- Utilisation d'un dépôt GitHub contenant des composants JS.
- Extraction et filtrage selon qualité et pertinence.

3.3.2.3 Création Manuelle de Templates Complets

Objectif : Fournir des templates web complets et bien structurés.

Processus :

- Création manuelle de templates pour assurer :
  - Cohérence structurelle
  - Esthétique visuelle
  - Applicabilité concrète
- Intégration HTML + CSS + JS

Cette figure montre un aperçu de la structure du dataset.

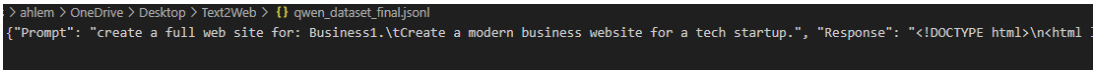

Figure 3.2 – Structure globale de projet

Caractéristiques Clés des Templates

- Basés sur des structures issues du monde réel.
- Couvrent toutes les technologies front-end.
- Variété : pages d'atterrissage, portfolios, formulaires de contact.
- Vise à enseigner la construction de pages web complètes.

3.3.3 Dataset d'exploration automatique de sites web

Dans le but de collecter du code HTML/CSS/JavaScript riche et réaliste, nous avons mis en place une méthode automatisée d'exploration de sites web à l'aide d'un outil de type *HackTrak* (outil de scraping avancé). Ce processus repose sur la récupération systématique du code source complet de sites proposant des templates web gratuits.

Méthodologie : Sélection des sources :
Une liste d'URLs de sites spécialisés dans les templates web gratuits (ex. : *HTML5 UP*, *Free CSS*, *TemplateMo*, etc.) a été définie comme point de départ.
Exploration automatique :
L'outil se charge d'ouvrir chaque lien, d'extraire le code source (HTML, CSS, JS) de la page principale, puis d'analyser tous les liens internes présents (menus, boutons, redirections, etc.).
Navigation récursive :
Pour chaque lien détecté, s'il mène à une page du même site ou à un sous-domaine connexe, l'outil poursuit l'extraction de manière récursive, jusqu'à l'absence de nouveaux liens internes ou un blocage d'accès (authentification, sécurité).
Organisation des fichiers :
Chaque site est enregistré dans un dossier structuré contenant :

- des fichiers HTML (`index.html`, `about.html`, `login.html`, etc.),
- les fichiers CSS (souvent regroupés dans un dossier `style/` ou `assets/`),
- les scripts JavaScript (`main.js`, `app.js`, etc.),
- et un dossier d'images et d'icônes.

Résultat : Plus de 500 sites web complets ont été extraits à l'aide de cette méthode, offrant une grande diversité de styles, de structures, et de logiques d'interaction frontend.

Limites rencontrées : Ce dataset n'a pas été exploité pour l'entraînement final, car :

- la taille des fichiers dépasse souvent `32K tokens`,
- notre matériel ne permettait pas de traiter efficacement ces longues séquences,
- même sur des plateformes cloud, les contraintes budgétaires rendaient l'exploitation de ce dataset trop coûteuse.

3.4 Entraînement

Pour la phase d'entraînement, nous avons utilisé la plateforme **RunPod**, un service cloud qui fournit un accès à des machines virtuelles dotées de GPU performants, adaptées aux tâches intensives en intelligence artificielle, notamment l'entraînement de modèles de génération de code.
Nous avons choisi une instance équipée d'un **GPU NVIDIA A100** avec 40 Go de mémoire et un espace disque de 40 Go, ce qui nous a permis de mener nos expériences dans des conditions optimales, en assurant un bon compromis entre puissance de calcul et coût.

3.4.1 Préparation de l'environnement

— **Étape 1 :** Sélection des options compatibles avec notre projet sur RunPod, ajout préalable du *token* Hugging Face, puis ouverture d'un environnement Jupyter Notebook (`.ipynb`).

— **Étape 2 :** Chargement des datasets et installation des bibliothèques essentielles voir (Figure 3.3) nécessaires à l'entraînement du modèle et à son chargement depuis Hugging Face.

```
!pip install transformers==4.51.1
!pip install datasets==3.5.0
!pip install accelerate==1.3.0
!pip install peft==0.14.0
!pip install huggingface-hub==0.30.2
!pip install tokenizers==0.21.0
!pip install safetensors==0.5.2
!pip install torch==2.5.1+cu124
!pip install numpy==1.26.4
!pip install pandas==2.2.3
!pip install tqdm==4.67.1
!pip install requests==2.32.3
!pip install pyarrow==19.0.1
```

FIGURE 3.3 – installation des bibliothèques nécessairet

— **Étape 3 :** pour le modèle, nous l'avons chargé directement depuis Hugging Face en utilisant un script Python contenant le token d'accès personnel.

3.4.2 Entraînement du modèle

3.4.2.1 Initialisation des paramètres

Dans cette section, les paramètres essentiels sont définis :

— `model_id` spécifie le nom du modèle pré-entraîné à utiliser depuis la plateforme Hugging Face.

— `dataset_path` correspond au chemin local du fichier `.jsonl` contenant les paires prompt/réponse à utiliser pour l'affinage du modèle.

— `max_tokens` détermine la taille maximale des séquences de tokens traitées par le modèle (ici, 8192 tokens).

```
model_id = "Qwen/Qwen2.5-Coder-0.5B-Instruct"
dataset_path = "qwen_dataset_final.jsonl"
max_tokens = 8192
```

FIGURE 3.4 – Initialisation des paramètres

3.4.2.2 Chargement du tokenizer

Le tokenizer est chargé depuis le modèle Qwen2.5-Coder (figure 3.5), permettant de transformer du texte brut en une séquence de tokens numériques exploitables par le modèle. On définit ici le token de padding comme étant identique au token de fin de séquence (`eos_token`) afin d'uniformiser les entrées.

```
tokenizer = AutoTokenizer.from_pretrained(model_id, trust_remote_code=True)
tokenizer.pad_token = tokenizer.eos_token
```

FIGURE 3.5 – Chargement de Tokenizer

3.4.2.3 Chargement et filtrage du dataset

La fonction `load_dataset` permet de charger les données depuis un fichier JSONL structuré avec les colonnes "Prompt" et "Response". L'ensemble est mélangé pour assurer l'aléa statistique dans l'entraînement.

3.4.2.4 Fonction de tokenisation

Cette fonction combine chaque prompt et sa réponse, puis applique une tokenisation avec troncature et remplissage automatique.
`input_ids` et `attention_mask` sont nécessaires à l'apprentissage, tandis que les labels (cibles d'apprentissage) sont identiques aux entrées (`input_ids`), comme c'est courant en modélisation causale (Causal Language Modeling).
`map` applique la tokenisation à tout le dataset.

```
def tokenize(example):
    text = example["Prompt"] + "\n" + example["Response"]
    tokens = tokenizer(text, truncation=True, padding="max_length", max_length=max_tokens, return_tensors="pt")
    return {
        "input_ids": tokens["input_ids"][0],
        "attention_mask": tokens["attention_mask"][0],
        "labels": tokens["input_ids"][0]
    }
tokenized_dataset = raw_dataset.map(tokenize, remove_columns=["Prompt", "Response"])
quant_config = BitsAndBytesConfig(
    load_in_8bit=True,
    llm_int8_enable_fp32_cpu_offload=True,
)
```

Figure 3.6 – Fonction de tokenisation

3.4.2.5 Configuration de LoRA

LoRA (Low-Rank Adaptation) permet de n'entraîner qu'une partie réduite des poids du modèle, rendant l'apprentissage plus rapide et moins coûteux.

- `r` est le rang utilisé dans la factorisation des matrices internes
- `lora_alpha` et `lora_dropout` influencent l'amplitude et la régularisation
- Les modules cibles désignent les couches internes du transformeur à adapter (principalement les couches d'attention et de projection)

```
model = prepare_model_for_kbit_training(model)
model.gradient_checkpointing_enable()
lora_config = LoraConfig(
    r=32,
    lora_alpha=16,
    lora_dropout=0.5,
    bias="none",
    target_modules=[
    "q_proj",
    "k_proj",
    "v_proj",
    "o_proj",
    "gate_proj",
    "up_proj",
    "down_proj"
],
    task_type="CAUSAL_LM"
)
model = get_peft_model(model, lora_config)
```

FIGURE 3.7 – Configuration du LoRA

3.4.2.6 Définition des paramètres d'entraînement

La classe `TrainingArguments` (figure 3.8) permet de spécifier tous les hyperparamètres nécessaires pour l'entraînement du modèle :

- `output_dir` : dossier où les résultats (checkpoints, logs, modèles) seront sauvegardés
- `per_device_train_batch_size=2` : taille du lot de données traité simultanément sur chaque GPU. Une valeur faible est utilisée ici pour limiter l'utilisation de mémoire
- `gradient_accumulation_steps=4` : les gradients sont accumulés sur 4 lots avant de mettre à jour les poids, ce qui simule un batch size total de $2 \times 4 = 8$
- `learning_rate=5e-4` : taux d'apprentissage initial ; une valeur modérée adaptée à un affinement en LoRA
- `num_train_epochs=1` : une seule époque est utilisée, ce qui peut suffire pour passer à toutes les lignes du dataset
- `warmup_steps=100` : nombre d'étapes pendant lesquelles le taux d'apprentissage

augmente progressivement pour stabiliser l'entraînement au début

— `logging_steps=30` : fréquence (en nombre de pas) d'enregistrement des métriques dans les logs

— `save_strategy="steps"` et `save_steps=500` : les checkpoints sont sauvegardés tous les 500 pas d'entraînement

— `save_total_limit=3` : seuls les 3 derniers checkpoints sont conservés pour économiser l'espace disque

— `fp16=True` : l'entraînement est effectué en précision mixte 16 bits, permettant une exécution plus rapide avec moins de mémoire

— `report_to="none"` : empêche l'envoi des logs à des outils de suivi externes (comme WandB ou TensorBoard)

— `logging_dir="/logs"` : dossier local pour stocker les journaux d'apprentissage

```
training_args = TrainingArguments(
    output_dir="/results",
    per_device_train_batch_size=2,
    gradient_accumulation_steps=4,
    learning_rate=5e-4,
    num_train_epochs=1,
    warmup_steps=100,
    logging_steps=30,
    save_strategy="steps",
    save_steps=500,
    save_total_limit=3,
    fp16=True,
    report_to="none",
    logging_dir="/logs"
)
```

Figure 3.8 – Paramètre d'entrainement

3.4.2.7 Création de l'objet Trainer

L'objet `Trainer` (figure 3.9) est une interface simplifiée pour la boucle d'entraînement. Il prend en entrée :

— Le modèle à affiner

— Les arguments d'entraînement définis précédemment
— Le jeu de données prétraité
— Le tokenizer, utile pour le traitement des données et la génération future

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset,
    tokenizer=tokenizer
)
```

FIGURE 3.9 – L'objet trainer

3.4.2.8 Lancement de l'entraînement

Après l'exécution de l'ensemble du code d'entraînement, l'état d'utilisation des ressources du pod (CPU, RAM, GPU, disque, etc.) est affiché comme illustré à la figure 3.12.

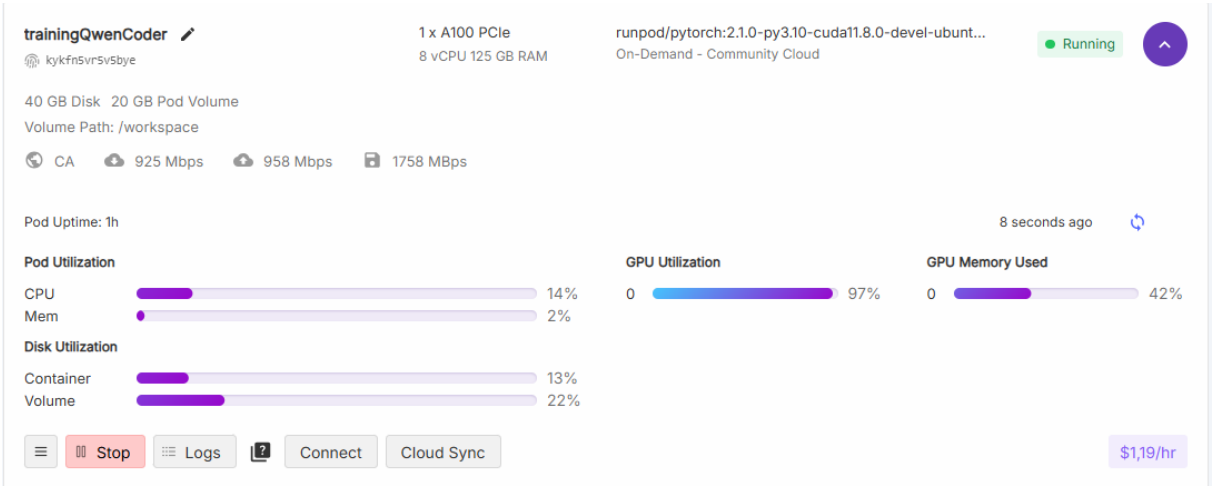

FIGURE 3.10 – L'état d'utilisation des ressources sur RunPod

Utilisation du processeur (CPU) et de la mémoire (RAM)

— **CPU** : 14 % d'utilisation, indiquant que l'entraînement est principalement géré par le GPU, avec une faible charge sur le processeur
— **Mémoire (RAM)** : 2 % utilisée, ce qui est très faible, confirmant que la charge mémoire est principalement déportée vers la VRAM du GPU

3.4.2.9 Résultat d'entraînement

Au cours de l'entraînement du modèle Qwen2.5-Coder-0.5B-Instruct, nous avons observé une réduction significative de la fonction de perte (training loss), traduisant une bonne convergence du modèle. Dès les premières étapes, la perte passe de 5.1138 à 0.2414 entre les étapes 30 et 60, ce qui indique une phase d'ajustement rapide aux données d'entraînement. Par la suite, la décroissance de la perte se poursuit de manière plus modérée, atteignant 0.1479 à l'étape 330. Cette évolution témoigne d'une stabilisation progressive de l'apprentissage.
L'utilisation de la quantification en 8 bits (via `BitsAndBytesConfig`) combinée au réglage fin avec LoRA (Low-Rank Adaptation) et à l'accumulation de gradients a permis d'optimiser l'entraînement tout en réduisant la consommation mémoire, ce qui est particulièrement adapté aux environnements à ressources limitées. Ces résultats montrent que la configuration choisie est efficace pour adapter le modèle pré-entraîné à notre jeu de données spécifique.

Step	Training Loss
30	5.113800
60	0.241400
90	0.192400
120	0.182500
150	0.169100
180	0.175300
210	0.163100
240	0.150700
270	0.159500
300	0.155900
330	0.147900

Figure 3.11 – Tableau de "step" et "loss" après l'entrainement

3.4.2.10 Sauvegarde finale du modèle et du tokenizer

Une fois l'entraînement terminé, le modèle affiné et le tokenizer associé sont sauvegardés dans le dossier `final_model`.

```
model.save_pretrained("final_model")
tokenizer.save_pretrained("final_model")
```

```
[38]: ('final_model/tokenizer_config.json',
       'final_model/special_tokens_map.json',
       'final_model/vocab.json',
       'final_model/merges.txt',
       'final_model/added_tokens.json',
       'final_model/tokenizer.json')
```

Figure 3.12 – Sauvegarde finale de fine tune LoRA

3.5 Fonctionnement du modèle final

Le modèle développé est spécialisé dans la génération de pages web cohérentes, stylisées et contextuellement adaptées, à partir d'un **jeu de données personnalisé**. Cette spécialisation est obtenue grâce à une phase de fine-tuning du modèle de base.

- **Type de modèle** : Modèle de langage causal (Causal Language Model) basé sur l'architecture **Transformer**, affiné via la méthode **LoRA (Low-Rank Adaptation)**.
- **Rôle** : Affiner le comportement du modèle de base pour qu'il soit mieux adapté à la tâche spécifique de génération de code web.
- **Spécificités** :
  - Il ne s'agit pas d'un modèle complet, mais uniquement d'un ensemble de poids d'ajustement.
  - Ces poids sont des **adapteurs de faible rang (low-rank adapters)**, qui doivent être appliqués au modèle de base pour fonctionner correctement.

3.5.1 Pré-traitement du prompt

Avant d'être traité par le modèle, le prompt structuré, envoyé par l'utilisateur via le backend, subit un ensemble d'étapes préliminaires :

- **Tokenisation** : Le texte du prompt est transformé en une séquence de tokens numériques à l'aide du **tokenizer du modèle Qwen2.5 Coder**, basé sur Auto-

Tokenizer de la bibliothèque Hugging Face.
Exemple de prompt : *« Créer une page de contact avec un formulaire et une carte intégrée. »* devient : `[50256, 1702, 857, 120, 2135, 390, 1110, 952, 2519, 1432, 1587]`

3.5.2 Traitement par le modèle

Une fois la séquence tokenisée, elle est traitée par le modèle selon les étapes suivantes :

- **Embedding** : Chaque token est converti en un vecteur dense via une table d'embedding.
- **Position Embedding** : Des vecteurs de position sont ajoutés pour préserver l'ordre séquentiel du texte.
- **Passage dans les blocs Transformer** : La séquence passe par **24 couches** successives, chaque couche comprenant :
  - Mécanisme d'**attention multi-têtes**
  - **Feedforward layers**
  - **Normalisation**
  - **Dropout**
- **Décodage auto-régressif** : Le modèle prédit un token à la fois en utilisant tous les tokens précédents (causalité). Il commence à générer à partir du token `<html>` et poursuit jusqu'à la fin du code (HTML, CSS, JS).

3.5.3 Spécificités du fine-tuning

- **Structure des données d'entraînement** : Les données ont été préparées sous forme structurée, incluant un prompt et un code complet :
  `<prompt>Créer une page contact ...</prompt>, <html>...</html>, <css>...</css>, <script>...</script>`
- **Type d'apprentissage** : Le fine-tuning a été réalisé en mode **apprentissage supervisé**, où :
  - Le prompt est fourni en tant qu'entrée (input)
  - Le code HTML, CSS et JS concaténé est la **séquence cible (output)**
  - Le modèle apprend à prédire chaque token du code à partir du contexte donné.

3.5.4 Sortie du modèle

- **Décodage** : La séquence de tokens générée est convertie en texte brut (HTML, CSS, JS) à l'aide de la fonction `tokenizer.decode()`.

3.6 Conclusion

En conclusion, nous avons réussi à construire des jeux de données adaptés et à optimiser le modèle Qwen2.5-Coder à l'aide d'un fine-tuning LoRA, ce qui a permis d'améliorer significativement la génération de sites web en termes de structure, d'esthétique et de pertinence par rapport au modèle de base. Ce travail constitue une base solide pour de futures avancées dans l'automatisation du développement web.Dans le chapitre suivant, nous procéderons à une évaluation approfondie du modèle final, puis nous présenterons l'interface utilisateur que nous avons développée pour interagir avec ce modèle.

Chapitre 4

Évaluation et résultats

4.1 Introduction

Dans ce chapitre, nous présentons les composantes essentielles de notre solution de génération automatique de sites web, en mettant en avant les méthodes d'évaluation mises en œuvre (évaluation automatique, tests A/B et évaluation humaine) ainsi que l'interface utilisateur, conçue pour une interaction fluide et intuitive. Nous incluons également une étude de marché visant à analyser les attentes, les comportements et les besoins des utilisateurs potentiels. Enfin, nous proposons des pistes d'amélioration pour guider les évolutions futures du système.

4.2 Évaluation

4.2.1 Évaluation Automatique

Définition de l'évaluation automatique : Processus de mesure et d'analyse des performances d'un modèle d'IA utilisant des métriques quantitatives et des algorithmes, sans intervention humaine directe. Elle permet d'évaluer rapidement et objectivement la qualité des résultats générés par le modèle selon des critères prédéfinis [50].

4.2.1.1 Évaluation avec des métriques NLP

Métrique	BLEU	ROUGE-1	ROUGE-2	ROUGE-L	Similarité Cosinus	CodeBERTScore
Score	0.2827	0.3191	0.1181	0.2713	0.5948	0.9650

Tableau 4.1 – Tableau 1 : Résultats d'évaluation avec les métriques NLP

Analyse des Résultats

Métriques Textuelles (BLEU/ROUGE)
Les scores BLEU (0.28) et ROUGE relativement modestes s'expliquent par la nature créative de la génération de code web. Contrairement à la traduction où une correspondance exacte est attendue, notre modèle prend des décisions stylistiques autonomes tout en respectant les contraintes techniques.
ROUGE-2 particulièrement faible (0.12) indique que le modèle génère des combinaisons de mots différentes de la référence, ce qui est normal pour un code créatif où plusieurs implémentations valides existent.
Métriques Sémantiques
La **similarité cosinus** (0.59) démontre une compréhension sémantique correcte des exigences, tandis que le **CodeBERTScore** exceptionnel (0.97) confirme que la structure et la fonctionnalité du code généré sont excellentes.
Limitations des Métriques Traditionnelles :
Les métriques BLEU/ROUGE, conçues pour l'évaluation textuelle, ne reflètent pas adéquatement la qualité du code généré car :

- **Créativité stylistique** : Le modèle choisit des approches CSS/JS différentes mais équivalentes
- **Flexibilité architecturale** : Plusieurs structures HTML valides pour un même objectif
- **Optimisation personnalisée** : Le modèle adapte le code selon le contexte du prompt

4.2.1.2 Évaluation comparative entre le modèle de base et le fine tune LoRA

Méthode d'Évaluation : L'évaluation compare deux modèles de langage pour la génération de pages web : un modèle de base (Qwen2.5-Coder 0.5B-Instruct) et sa version fine-tunée avec des adaptateurs LoRA. Le processus utilise un prompt spécifique : "Create an artist website. Design an indie artist website with soft pastel aesthetics, a blog section for behind the-scenes stories, and a playlist."
Les modèles génèrent des pages HTML complètes incluant CSS et JavaScript.
Génération des Pages Web : Chaque modèle génère une page HTML pour le prompt donné.
Métriques d'Évaluation :

- Structure et Qualité : Analyse des balises HTML, des règles CSS, des fonctions JavaScript, et des balises sémantiques.
- Pertinence : Mesure de la correspondance entre le prompt et le contenu généré.
- Richesse et Complétude : Évaluation de la longueur du HTML, de la complétude

structurelle (présence de DOCTYPE, head, body, etc.), et de la richesse du contenu.

Comparaison : Les performances des deux modèles sont comparées via des métriques quantitatives et visualisées avec des graphiques.

Résultats et Analyse :

- Longueur du HTML (total_length) : Le modèle fine-tuné produit un HTML beaucoup plus long (3601 caractères), indiquant une génération plus détaillée.
- Balises HTML et Sémantiques : Le modèle fine-tuné génère 42 balises HTML, dont 5 sémantiques (comme <header> ou <section>), contre aucune pour le modèle de base, montrant une structure plus riche et moderne.
- Règles CSS : Le modèle fine-tuné inclut 16 règles CSS, contre 0 pour le modèle de base, ce qui améliore l'esthétique et la responsivité.
- Complétude Structurelle et Richesse : Le modèle fine-tuné atteint un score parfait (1.0) pour la complétude structurelle (présence de DOCTYPE, head, body, etc.) et la richesse du contenu.

Graphiques : Les graphiques (boxplots et barres) de la (Figure 4.1) montrent une supériorité claire du modèle fine-tuné sur toutes les métriques, notamment en termes de structure, richesse, et pertinence.

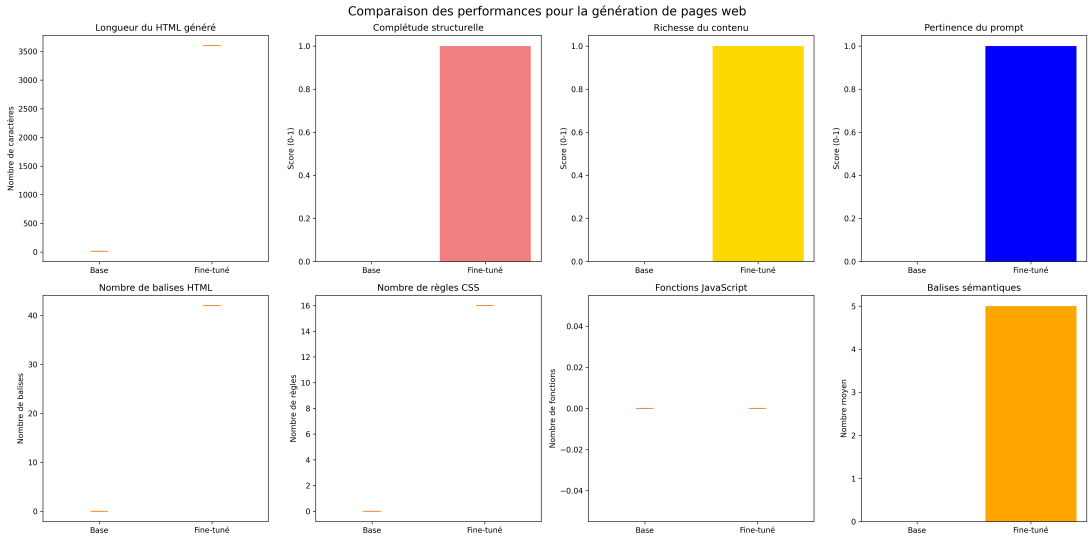

FIGURE 4.1 – Graphiques comparatives des performance entre le modèle de base et fine tune

En résumé : Le modèle fine-tuné surpasse largement le modèle de base, avec une amélioration significative dans la génération de pages web fonctionnelles et esthétiques. Il répond mieux aux exigences du prompt, produit un code plus structuré et visuellement attrayant, et intègre des fonctionnalités interactives.

4.2.2 Évaluation de type A/B

Définition de l'évaluation A/B : L'évaluation A/B consiste à comparer deux versions d'un modèle ou d'un système afin de déterminer laquelle donne les meilleurs résultats dans un même contexte [55].

Nous avons utilisé le même prompt pour les deux modèles afin de comparer leurs performances de manière équitable :

> *"Create an artist website. Design an indie artist website with soft pastel aesthetics, a blog section for behind-the-scenes stories, and a playlist."*

Pour le modèle de base, nous avons obtenu la page suivante :

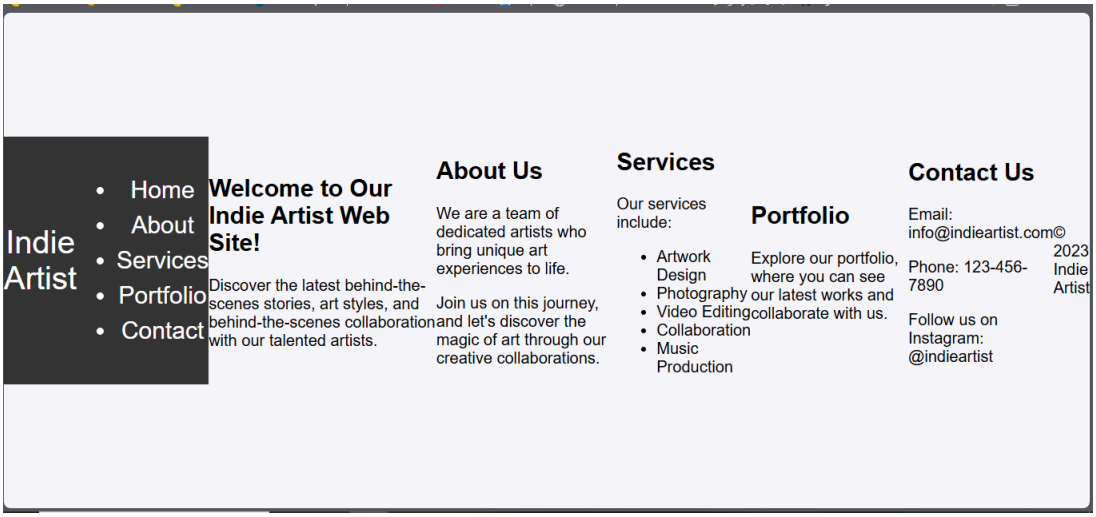

FIGURE 4.2 – Site web généré par le modèle de base seul

Remarque : Comme l'illustre l'image, la page web générée présente une qualité inférieure aux attentes. Le style est minimaliste, et la disposition des éléments HTML manque de structure et de cohérence visuelle.

Ensuite, nous avons exécuté le deuxième script pour tester le modèle fine-tuné sur le même prompt.

Pour ce modèle, nous avons obtenu une page web contenant des éléments plus cohérents avec la demande.

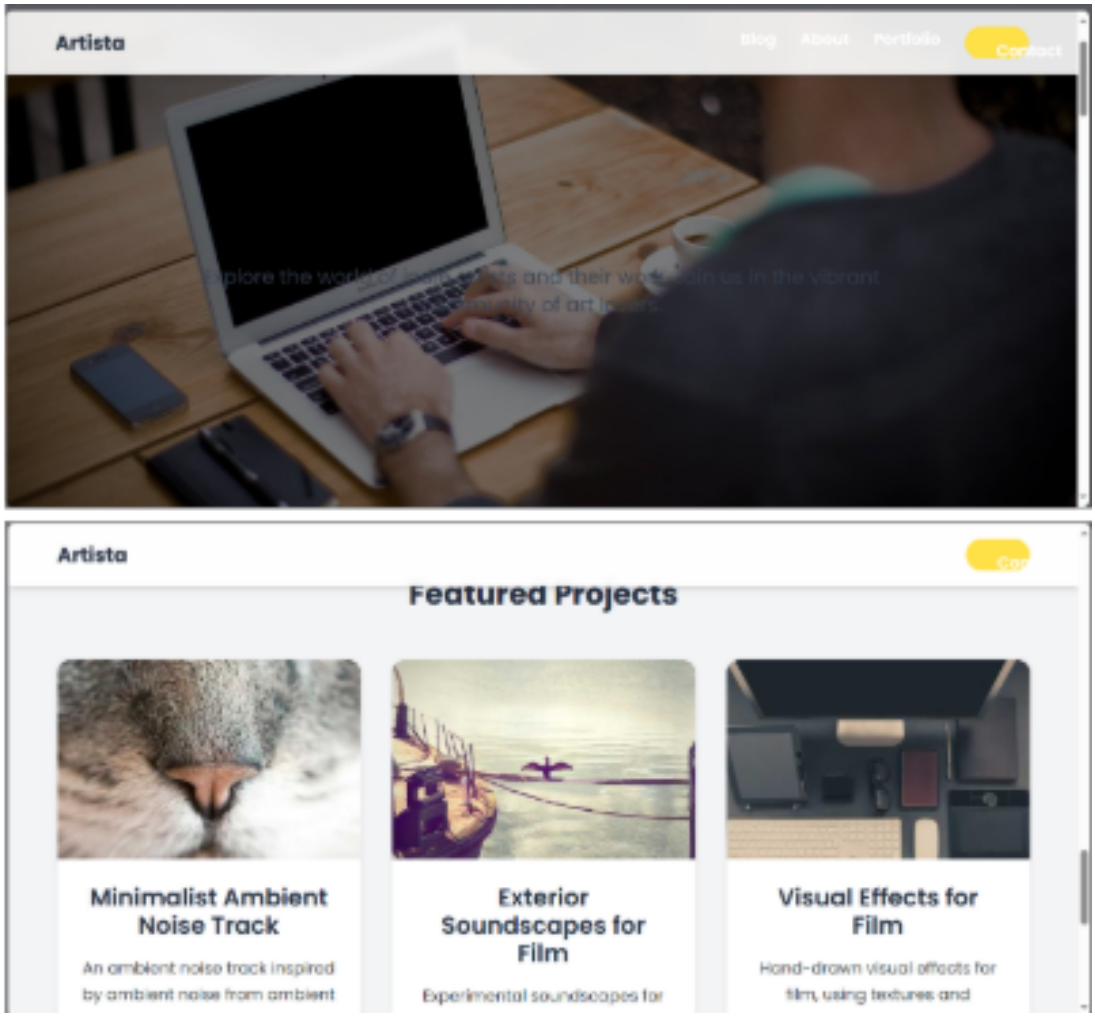

Figure 4.3 – Site web généré par le modèle fine tuné

En revanche, le modèle fine-tuné a produit une page web nettement plus aboutie. Le style est visuellement riche et soigné, avec une palette de couleurs harmonieuse correspondant à l'esthétique évoquée dans le prompt. Les éléments HTML sont bien structurés et organisés, offrant une navigation claire et agréable.

De manière notable, le modèle a même intégré des images, bien que le prompt ne les ait pas explicitement mentionnées. Cette capacité à enrichir le contenu de manière contextuelle démontre une meilleure compréhension des attentes implicites, ainsi qu'une amélioration significative des performances après le fine-tuning.

4.2.3 Évaluation humaine

Définition : C'est une analyse réalisée par une personne (et non une machine) pour juger la qualité d'un contenu selon des critères comme le design, la clarté, la pertinence,

ou l'expérience utilisateur [56].

Évaluation :

Utilisation du même prompt : *"Create an artist website. Design an indie artist website with soft pastel aesthetics, a blog section for behind-the-scenes stories, and a playlist."*

Critère	**Évaluation**	**Note**
Thème (catégorie) "artiste"	Le nom "Artista" et les sections sont cohérentes pour un site artistique.	4/4
Esthétique pastel douce	Les couleurs définies (#00BFFF, #FFC107, #FEE047) sont vives mais pas très *pastel*.	2/4
Section Blog	Une section blog est présente et bien stylisée.	4/4
Contenu "Behind the scenes"	Il manque du contenu textuel simulé pour donner l'effet "coulisses"	2/4
Présence de Playlist	Aucune trace d'un lecteur audio ou d'intégration musicale.	0/4

Tableau 4.2 – Respect du Prompt

Critère	**Évaluation**	**Note**
Structure HTML propre	Bonne structuration avec balises sémantiques (header, section, etc.).	4/4
Organisation du CSS	Utilisation claire des variables, bon respect des conventions.	4/4
Responsivité	Bonne gestion des media queries et Tailwind.	4/4
Séparation des styles	Fichier CSS externe bien mentionné	4/4
Accessibilité / SEO de base	Pas de balises ARIA ou attributs alt complets visibles.	1/4

Tableau 4.3 – Qualité du Code

Critère	Évaluation	Note
Identité visuelle cohérente	Bonne typographie, bon branding avec "Artista".	4/4
Navigation fluide	Menu responsive, liens clairs.	4/4
Palette de couleurs pastel	Plutôt flashy que pastel, pourrait être plus douce.	3/4
Animation et interactivité	Hover effets élégants, transitions douces.	4/4
Modernité du design	Design actuel, bien aligné avec les tendances.	4/4

Tableau 4.4 – Esthétique & UX

Critère	Évaluation	Note
Texte présent	Peu de contenu textuel (ex : "about", "blog" trop génériques).	1/4
Personnalisation artistique	Peu d'éléments spécifiques à un artiste (nom, style, biographie).	1/4
Multimédia	Images présentes mais pas de vidéos ou audios.	2/4
Playlist ou audio	Manquant, ce qui était une partie importante du prompt.	0/4
Qualité du contenu affiché	Bon design visuel mais manque de texte rédigé humainement.	4/4

Tableau 4.5 – Contenu

Critère	Évaluation	Note
Menu responsive	Fonctionnel avec icône burger.	4/4
Liens de navigation	Bien liés aux sections (Blog, About, Portfolio).	4/4
Intégration d'animations	Utilisation de keyframes personnalisées dans Tailwind.	3/4
Composants interactifs	Blog-posts animés, hover, mais pas de formulaire ou lecteur musical.	2/4
Contact ou interaction utilisateur	Bouton contact présent mais sans fonction associée.	1/4

Tableau 4.6 – Fonctionnalités

Note Globale : 70/100

Résumé : Le site généré est fonctionnel, moderne et bien structuré. Il reflète bien un style artistique, avec une bonne UX et des transitions douces. Toutefois, il manque des éléments essentiels demandés dans le prompt, notamment une playlist ou un lecteur audio, ainsi que des textes plus personnalisés pour l'artiste et les coulisses de création.

4.3 Interface Utilisateur

L'interface utilisateur propose une expérience simple, combinant génération automatique de code et modification manuelle. Elle se compose de deux pages principales :

Interface Principale

- **Navigation** : Barre d'état du serveur avec trois boutons : *Generate*, *Download*, *Modify*
- **Disposition** : Zone de saisie du prompt à gauche, affichage du code généré à droite
- **Utilisation** : L'utilisateur saisit un prompt, génère le code, puis peut le télécharger ou l'éditer

Éditeur Visuel (GrapesJS)

- **Interface** : Canevas central avec panneau de blocs à gauche
- **Fonctionnalités** : Glisser-déposer d'éléments, modification des styles, prévisualisation responsive
- **Sauvegarde** : Export du résultat final

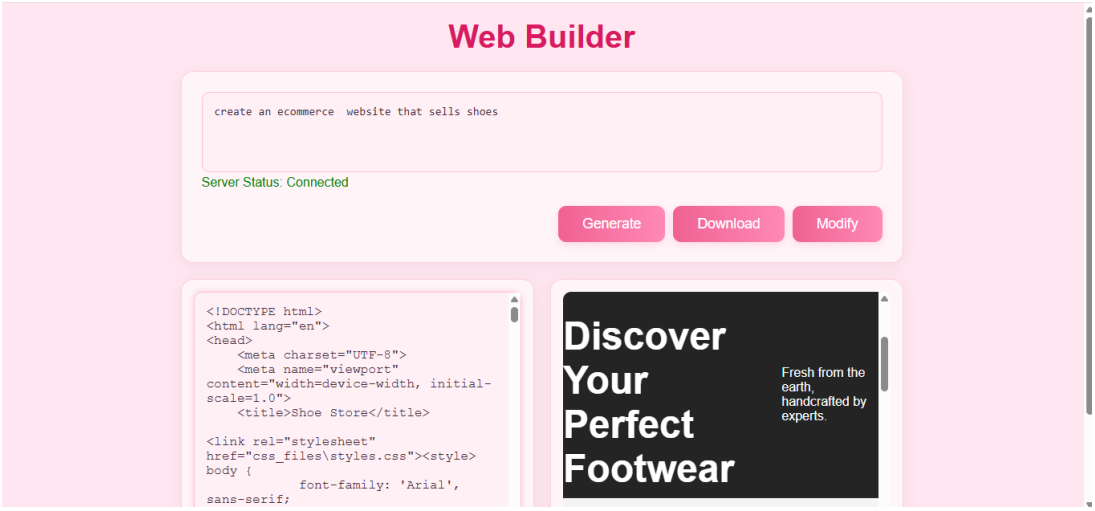

Figure 4.4 – Interface Utilisateur

4.3.1 Back-End

Architecture Le back-end repose sur **Flask** pour traiter les requêtes de génération de code.

- **Serveur** : Un endpoint /process_prompt reçoit les prompts
- **Modèle IA** : Utilisation de **Qwen0.5-instruct** avec adaptation **LoRA** pour produire du code HTML/CSS/JS
- **Optimisation** : Accélération via GPU pour réduire la latence

4.3.2 Éditeur GrapesJS

Présentation GrapesJS permet une édition visuelle intuitive des pages web :

- **Blocs** : Éléments prêts à l'emploi (texte, images, boutons, etc.)
- **Styles** : Modification directe des composants
- **Export** : Génération et sauvegarde du code HTML/CSS

Intégration

- **Importation** : Le code généré peut être importé dans l'éditeur
- **Persistance** : Sauvegarde automatique via le stockage local
- **Workflow** : Génération par IA → Import dans l'éditeur → Modification → Export final

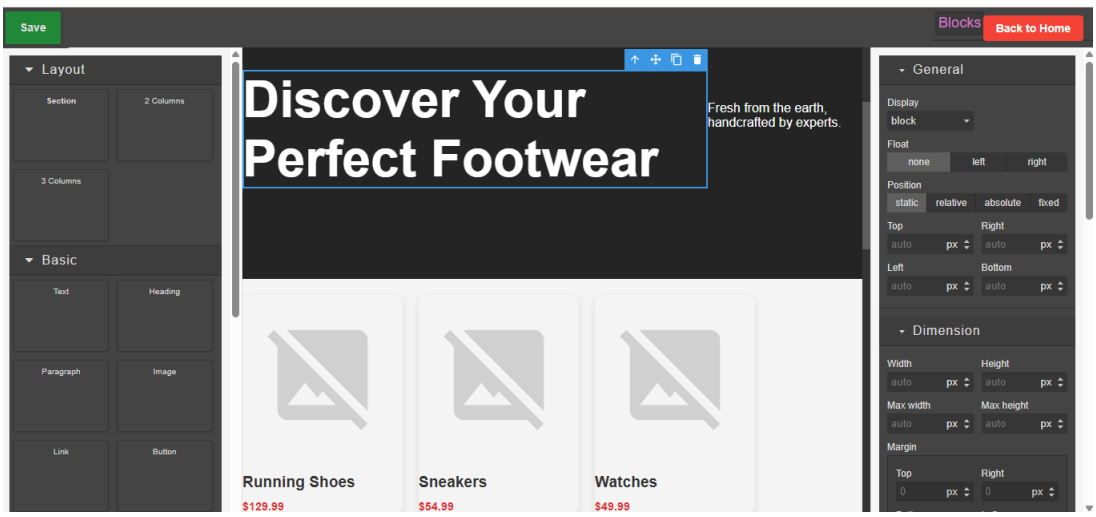

FIGURE 4.5 – Interface GrapesJS

4.3.3 Flux global du système

Le diagramme ci-dessous(Figure 4.6) résume le flux de travail de l'interface. Il illustre les différentes étapes du processus, depuis la saisie du prompt utilisateur jusqu'à l'édition

visuelle de la page générée.

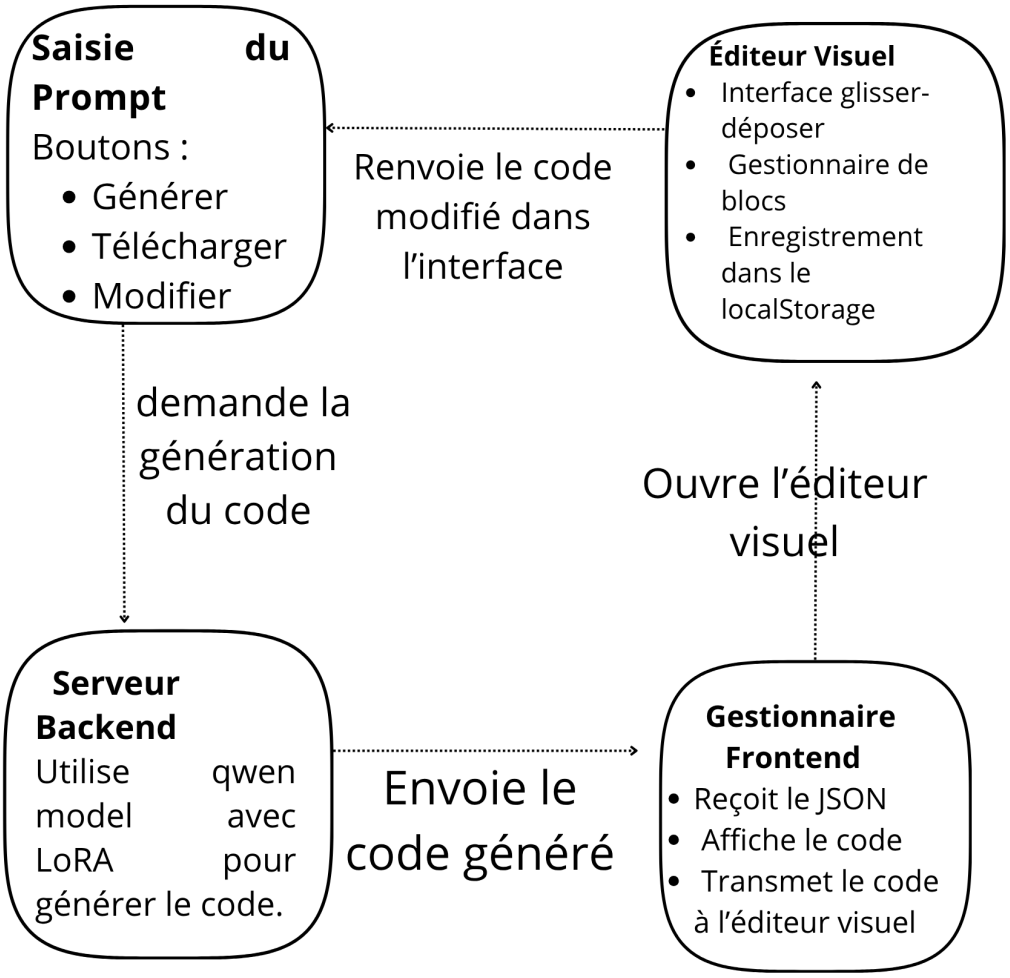

FIGURE 4.6 – Flux de système

4.4 Comparaison entre les systèmes existants

Cette section présente une comparaison entre les solutions similaires existantes. L'objectif est d'identifier leurs points forts et faibles, afin de situer notre contribution dans le paysage actuel des outils de génération de code.

Nom du système	**Avantages**	**Inconvénients**	**Détails**	**Open Source**	**Tarification**	**Génère un site par prompt**
Wix ADI	Facile à utiliser - Adapté aux débutants - Créer des sites à partir de questions	Limité aux sites statiques. Pas idéal pour des développeurs cherchant à coder	Utilise des algorithmes d'IA pour générer des sites web visuels basés sur des réponses utilisateur. Permet de choisir un modèle pour la personnalisation. Interface simple, sans compétences techniques nécessaires	Non	Free Premium	Oui, Prompt simple
Wix Studio	Designs responsifs pour mobile et tablette - modèles intuitifs	Édition détaillée parfois déroutante. Coût élevé	Permet de créer un site avec des mises en page responsives. Idéal pour des designs visuellement attrayants. Personnalisation approfondie avec une interface créative	Non	Free Premium	Non, Conception manuelle assistée

GoDaddy AI Builder	Facile à configurer - Interface conviviale	Limité aux sites - Moins flexible pour des designs sur mesure - Limité aux utilisateurs GoDaddy	Créer un site rapide à partir de réponses aux questions. Idéal pour les petites entreprises et les utilisateurs novices. Personnalisation simple mais avec moins de flexibilité	Non	Free Premium	Oui, Questions guidées
Zyro by Hostinger	Intégrations ecommerce - Outils de création de contenu	Personnalisation avancée limitée. Nécessite un abonnement	Génère des sites web avec des fonctionnalités dynamiques telles que l'e-commerce. Personnalisation limitée dans le cadre des plans gratuits. Conçu pour les petites entreprises avec un focus sur le design	Non	Payant	Oui, Similaire à Wix ADI avec prompt simplifié

Joomla	Multilingue ; système de templates robuste ; extensions nombreuses	Courbe d'apprentissage légère ; nécessite plus de configuration	CMS flexible et personnalisable. Idéal pour les sites à contenu lourd. Moins adapté aux utilisateurs novices à cause de la courbe d'apprentissage. Support multilingue natif	Oui	Gratuit	Non, CMS manuel (pas de génération IA)
Ghost	Interface propre, technologie moderne ; personnalisé avec des thèmes	Conçu principalement pour les blogueurs ; moins adapté pour les sites à fonctionnalités complexes	CMS minimaliste, parfait pour les blogueurs et rédacteurs. Idéal pour les sites à contenu textuel. Facile à personnaliser avec des thèmes. Moins adapté aux sites d'e-commerce	Oui	Free Premium	Non, CMS pour blogs, sans IA
PrestaShop	Solution ecommerce complète ; passerelles de paiement multiples	Nécessite de la configuration et de l'expertise technique pour une personnalisation poussée	Flexible avec un large éventail de modules. Bonne option pour créer des boutiques en ligne professionnelles	Oui	Gratuit	Non, CMS e-commerce, manuel

Tableau : Comparaison des systèmes existants

Les systèmes mentionnés ci-dessus sont soit payants, donc non accessibles à tout le monde, soit leurs versions gratuites sont très limitées. De plus, certains d'entre eux sont relative-

ment complexes à utiliser et nécessitent des formations pour apprendre à les maîtriser, ce qui les rend réservés à des utilisateurs expérimentés. D'autres ne génèrent pas de code, mais permettent plutôt à l'utilisateur de construire son site par un système de glisser-déposer, un peu comme WordPress.
Ainsi, notre système se distingue par sa simplicité : l'utilisateur saisit simplement une phrase décrivant le site qu'il souhaite, dans n'importe quel domaine, et obtient un résultat en quelques secondes, gratuitement. Il peut ensuite effectuer les modifications qu'il souhaite.

4.5 Conclusion

En conclusion, nous avons développé une solution intégrée combinant génération automatique de sites web et édition manuelle, grâce à une interface intuitive, un back-end Flask optimisé, et GrapesJS. Le modèle Qwen2.5-Coder, fine-tuné avec LoRA, a permis de produire des pages fonctionnelles et esthétiques. Ce chapitre a démontré l'efficacité de notre système tout en ouvrant la voie à des évolutions futures, notamment vers la création de sites dynamiques plus complexes.

Conclusion Générale

L'intelligence artificielle redéfinit profondément notre rapport à la technologie. En simplifiant, automatisant et enrichissant des tâches autrefois réservées aux experts, elle rend l'innovation plus accessible que jamais. Ce mémoire s'inscrit pleinement dans cette dynamique, en explorant comment l'IA générative peut transformer la conception et le développement de sites web.

Nous avons démontré qu'il est désormais possible, grâce à des modèles d'apprentissage automatique, de passer d'un simple prompt en langage naturel à une page web statique, fonctionnelle et soignée. En combinant un modèle affiné (Qwen2.5-Coder), une interface utilisateur claire et un éditeur visuel intégré, notre système permet même aux non-développeurs de créer facilement un site web.

Cependant, ce travail n'a pas été sans défis. Nous avons été confrontés à un manque de ressources matérielles adaptées, ce qui a limité la vitesse et l'efficacité de nos phases d'entraînement et de test. De plus, la collecte du dataset a représenté un véritable défi pour nous, notamment en raison de l'absence de dataset prêt à l'emploi adapté à notre cas d'usage.

Mais ce projet n'est qu'un point de départ. À l'avenir, nous souhaitons développer une plateforme plus complète capable de générer des sites web dynamiques multi-pages, incluant la conception de bases de données, l'intégration backend avec SQL et PHP, et la gestion de contenu dynamique. L'objectif est de créer une solution intelligente qui, à partir d'un simple prompt, produit un site fonctionnel, adapté à divers domaines, avec un minimum d'intervention humaine. Cette plateforme pourrait devenir un assistant virtuel complet pour le développement web.

Bibliographie

[1] M. I. Jordan and T. M. Mitchell. Machine learning : Trends, perspectives, and prospects. *Science*, 349(6245) :255–260, 2015.

[2] Auteur anonyme. Applications of machine learning and deep learning in aerospace engineering and aero-engine engineering. Technical report, Rapport de recherche, 2024.

[3] Dr. Zahra. What is artificial intelligence, machine learning and deep learning. Cours dispensé dans le module Advanced Topics in Computer Systems and Networks, Master 2 Computer & Network Engineering, Université de Blida 1, Blida, Algérie, 2025.

[4] W. Zhang, H. Li, Y. Li, et al. Application of deep learning algorithms in geotechnical engineering : a short critical review. *Artificial Intelligence Review*, pages 1–41, 2021.

[5] H. Bouhaouel. Chapitre ii : Réseaux de neurones artificiels. Master's thesis, Université Ibn Khaldoun – Tiaret, 2023. Mémoire de Master.

[6] Y. Ben Youcef. Les réseaux de neurones. Master's thesis, Université Kasdi Merbah Ouargla, 2020. Mémoire de Master académique en Génie Chimique, année universitaire 2019/2020.

[7] Botpress. Qu'est-ce qu'un réseau neuronal profond ? Technical report, Botpress, 2024.

[8] S. S. Sengar, A. B. Hasan, S. Kumar, and F. Carroll. Generative artificial intelligence : A systematic review and applications. arXiv preprint arXiv :2405.11029, 2024.

[9] AI-PRO Team. Generative ai : Here to stay, but for good ? Technical report, AI-Pro, October 2024.

[10] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851, 2020.

[11] A. Gu and T. Dao. Mamba : Linear-time sequence modeling with selective state spaces. arXiv preprint arXiv :2312.00752, 2023.

[12] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8) :9, 2019.

[13] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. Hierarchical text-conditional image generation with clip latents. arXiv preprint arXiv :2204.06125, 2022.

[14] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. The curious case of neural text degeneration. arXiv preprint arXiv :1904.09751, 2019.

[15] A. Fan, M. Lewis, and Y. Dauphin. Hierarchical neural story generation. arXiv preprint arXiv :1805.04833, 2018.

[16] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian. A comprehensive overview of large language models. arXiv preprint arXiv :2307.06435, 2023.

[17] SciSpace. Large language models in nonprofit and public good sectors. Technical report, 2024.

[18] Q. Niu, J. Liu, Z. Bi, P. Feng, B. Peng, K. Chen, and M. Li. Large language models and cognitive science : A comprehensive review of similarities, differences, and challenges. Technical report, 2024.

[19] Qwen Team. Qwen2.5-coder technical report. arXiv preprint arXiv :2409.12186, 2024.

[20] S. Wang et al. Recode : Robustness evaluation of code generation models. arXiv preprint arXiv :2212.10264, 2022.

[21] X. Li and Z. Wang. Generative ai : Principles and applications. *Artificial Intelligence Review*, 56(7) :789–812, 2023.

[22] Glacier Media Digital. Unlocking the power of online presence : Why 24/7 accessibility matters. Technical report, March 2024.

[23] C. Chakir. Créateur de site web vs codage : Quelle méthode vous convient le mieux ? Technical report, Hostinger, July 2024.

[24] I. Gur, O. Nachum, Y. Miao, M. Safdari, A. Huang, A. Chowdhery, S. Narang, N. Fiedel, and A. Faust. Understanding html with large language models. arXiv preprint arXiv :2210.03945, 2023.

[25] N. A. Ikumapayi and B. D. Oladokun. Automated front-end code generation using openai : Empowering web development efficiency. SSRN, 2023. DOI : 10.2139/ssrn.4590704.

[26] J. Ji. Experiments on code generation for web development using llms : A comparative study. Thèse de licence, Università di Bologna – Campus di Cesena, 2024. Département d'informatique – Science et ingénierie. Relatore : Prof.ssa Silvia Mirri.

[27] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y.K. Li, F. Luo, Y. Xiong, and W. Liang. Deepseek-coder : When the large language model meets programming - the rise of code intelligence. arXiv preprint arXiv :2401.14196, 2024.

[28] R. Hosseini and P. Rydén. Generative ai in frontend web development : A comparison between ai as a tool and as a sole programmer. Degree project, KTH Royal Institute of Technology, 2024.

[29] Amazon Web Services. Qu'est-ce que l'ingénierie du prompt ? – explication de l'ingénierie du prompt liée à l'ia. Technical report, 2025. consulté le 9 juin 2025.

[30] Analytics Vidhya. Mastering decoder-only transformer : A comprehensive guide. Technical report, 2024.

[31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[32] B. Bo. 3 pitfalls of rnns and why transformers rule the world. Medium, May 2021.

[33] B. Peng, S. Narayanan, and C. Papadimitriou. On limitations of the transformer architecture. Technical report, 2024.

[34] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang. Pre-trained models for natural language processing : A survey. *Science China Technological Sciences*, 63(10) :1872–1897, 2020.

[35] J. Howard and S. Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*, pages 328–339, 2018.

[36] M. E. Peters, M. Neumann, R. Logan, R. Schwartz, V. Joshi, S. Singh, and N. A. Smith. Knowledge enhanced contextualized word representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pages 43–54, 2019.

[37] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, and J. Kang. Biobert : a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4) :1234–1240, 2020.

[38] M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks : The sequential learning problem. *Psychology of Learning and Motivation*, 24 :109–165, 1989.

[39] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, others, and S. Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799, 2019.

[40] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, others, and W. Chen. Lora : Low-rank adaptation of large language models. arXiv preprint arXiv :2106.09685, 2021.

[41] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer. Qlora : Efficient finetuning of quantized llms. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[42] X. Liu, Y. Zheng, Z. Du, M. Ding, Y. Qian, Z. Yang, and J. Tang. Gpt understands, too. *AI Open*, 3 :40–53, 2022.

[43] J. He, C. Zhou, X. Ma, T. Berg-Kirkpatrick, and G. Neubig. Towards a unified view of parameter-efficient transfer learning. arXiv preprint arXiv :2110.04366, 2021.

[44] X. Liu, K. Ji, Y. Fu, W. L. Tam, Z. Du, Z. Yang, and J. Tang. P-tuning v2 : Prompt tuning can be comparable to fine-tuning universally across scales and tasks. arXiv preprint arXiv :2110.07602, 2021.

[45] OpenAI. Gpt-4 technical report. arXiv preprint arXiv :2303.08774, 2024.

[46] Akmmus AI. A survey of gpt-3 family large language models including chatgpt and gpt-4. arXiv preprint arXiv :2310.12321, 2023.

[47] Ce Zhou, Qian Li, Chen Li, Jun Yu, Yixin Liu, Guangjing Wang, Kai Zhang, Cheng Ji, Qiben Yan, Lifang He, Hao Peng, Jianxin Li, Jia Wu, Ziwei Liu, Pengtao Xie, Caiming Xiong, Jian Pei, and Philip S. Yu. A comprehensive survey on pretrained foundation models : A history from bert to chatgpt. arXiv preprint arXiv :2302.09419, 2023.

[48] Y. Wang, Q. Li, J. Zhao, M. Zhang, Y. Liu, and Y. Zhang. Codegemma : A family of code models built on gemma. arXiv preprint arXiv :2409.12186, 2024.

[49] Q. Chen, Y. Li, J. Li, and J. Ma. Wizardcoder-python-34b-v1.0 : Empowering python coding with large language models. arXiv preprint arXiv :2505.03733, 2025.

[50] Alibaba Cloud Community. Introducing qwen2.5 coder 32b instruct | qwen. Technical report, 2025. consulté en juin 2025.

[51] Innovatiana. Techniques d'évaluation des modèles de machine learning. Technical report, 2024.

[52] DataCamp. Qwen coder 2.5. Technical report, 2025. consulté en juin 2025.

[53] Qwen Team. Qwen2.5 technical report (version 2). arXiv preprint arXiv :2412.15115, 2025.

[54] Qwen Team. Qwen3 technical report (version 1). arXiv preprint arXiv :2505.09388, 2025.

[55] D. Wei. Demystifying a/b testing in machine learning. Presented in the Journal of Applied Machine Learning Research, 2025. Vol. 12, No. 1, pp. 45–58.

[56] S. Parker. The crucial role of human evaluation in prompt engineering for ai success. Presented at the International Conference on Human-Centered Artificial Intelligence (HCAI), 2024. pp. 110–120.

[57] AI-PRO Team. A definitive answer to "can i generate code using generative ai ?". Technical report, AI-Pro, October 2024.

[58] M. Chen, L. Zhou, Z. Liu, J. Shi, W. Yang, H. Li, others, and Z. Wei. Deepseek-vl : Towards language and vision foundation model via mixture of experts. arXiv preprint arXiv :2402.06196, 2024.

[59] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, volume 30, 2017.

Liens et dépôts en ligne :

Interactive Web Projects[1], Build A Quiz App[2], 50Projects - AdsBlockerExtension[3], Creative Loading Animation[4], 100 Mini Projects in HTML CSS JS[5], UI Reasoning Dataset[6]