UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE DU PROGRAMME DE MAÎTRISE EN MATHÉMATIQUES ET INFORMATIQUE APPLIQUÉES

BÂTIMENTS AUTONOMES, INTELLIGENTS ET COMMUNICANTS À BASE DE MODÈLES DE LANGAGE (LLMs) : ÉTUDE DU CAS NaviUQTR

PRÉSENTÉ PAR

SALIOU DIA

DÉPARTEMENT DE MATHÉMATIQUES ET D'INFORMATIQUE

AVRIL 2026

Résumé

La navigation intérieure dans les environnements universitaires complexes représente un défi persistant pour les usagers, particulièrement les nouveaux arrivants confrontés à une signalétique insuffisante et à des codes de salles peu intuitifs. Les systèmes traditionnels reposent sur des interfaces rigides nécessitant la connaissance préalable de codes alphanumériques, limitant leur accessibilité. Ce mémoire propose NaviUQTR, un système de navigation intérieure intelligent capable d'interpréter des requêtes en langage naturel et de générer des itinéraires personnalisés avec instructions contextualisées.

L'approche développée articule une modélisation spatiale sous forme de graphe topologique multiniveau, un module d'identification basé sur l'affinage du modèle Mistral-7B avec la technique LoRA, un générateur d'instructions verbales, et une interface utilisateur interactive. La contribution méthodologique majeure réside dans une stratégie itérative d'amélioration du corpus d'entraînement guidée par l'analyse des limitations observées. À travers trois versions successives, la composition du corpus a été optimisée en privilégiant l'équilibrage qualitatif plutôt que l'augmentation quantitative, améliorant la précision de 37,5% (V1) à 93,8% (V2.1).

Les résultats confirment la viabilité d'une approche hybride combinant planification algorithmique et compréhension linguistique. Le système atteint des performances quasi parfaites sur toutes les catégories de requêtes avec un ratio optimal de 1,10, ouvrant des perspectives prometteuses pour le déploiement de systèmes de navigation accessibles et adaptatifs.

Mots-clés : Navigation intérieure, Modèles de langage, Affinage, LoRA, Traitement du langage naturel, Planification de chemins

---

Abstract

Indoor navigation in complex university environments remains a persistent challenge for users, particularly newcomers facing insufficient signage and unintuitive room codes. Traditional systems rely on rigid interfaces requiring prior knowledge of alphanumeric codes, limiting accessibility. This thesis proposes NaviUQTR, an intelligent indoor navigation system capable of interpreting natural language queries and generating personalized routes with contextualized instructions.

The developed approach integrates spatial modeling as a multi-level topological graph, a destination identification module based on fine-tuning the Mistral-7B model with the LoRA technique, a verbal instruction generator, and an interactive user interface. The major methodological contribution lies in an iterative strategy for improving the training corpus guided by systematic analysis of observed limitations. Through three successive versions, corpus composition was optimized by prioritizing qualitative balance over quantitative increase, improving accuracy from 37.5% (V1) to 93.8% (V2.1).

Results confirm the viability of a hybrid approach combining algorithmic planning and linguistic understanding. The system achieves near-perfect performance across all query categories with an optimal validation/entraînement ratio of 1.10, opening promising perspectives for deploying accessible and adaptive navigation systems.

Keywords : Indoor navigation, Language models, Fine-tuning, LoRA, Natural language processing, Path planning

---

Dédicace

À la mémoire de mon cher père,

Dont l'amour, les conseils et la présence bienveillante continuent de guider chacun de mes pas. Tu m'as transmis le goût de l'effort, la rigueur intellectuelle et le sens du dépassement. Bien que ton absence physique pèse chaque jour, ton héritage spirituel demeure ma plus grande force.

Ce travail, fruit de longues heures de recherche et de réflexion, porte l'empreinte de tes enseignements. Tu m'as appris que la connaissance ne vaut que si elle sert le bien commun, que l'excellence exige humilité et persévérance, et que les obstacles ne sont jamais des fins en soi, mais des opportunités de grandir.

Que ce mémoire soit un hommage à ta sagesse, à tes sacrifices et à la lumière que tu as laissée en moi. Qu'Allah, le très miséricordieux, t'accorde sa grâce infinie et t'accueille dans son paradis éternel.

Je te dédie ce travail avec respect, gratitude et amour éternel.

Ton fils, Saliou

Remerciements

Je remercie d'abord Dieu, le Tout-Puissant, pour la santé, la patience et la persévérance qui m'ont permis de mener à terme ce mémoire.

J'exprime ma profonde gratitude à ma mère, Sokhna Diakhate, pour ses prières, son soutien constant et ses sacrifices innombrables. Tu as toujours cru en mes capacités, même lorsque les obstacles paraissaient insurmontables. Je garde également une pensée émue pour mon défunt père, dont les enseignements et la sagesse continuent de guider mes choix. Votre amour constitue le socle sur lequel j'ai bâti mes ambitions académiques.

Je tiens à remercier ma conjointe, Mame Diarra Bousso Ndiaye, pour son appui, sa compréhension et sa patience tout au long de mes études. Les longues heures de travail et les moments d'incertitude, tu as tout partagé avec une bienveillance qui m'a porté jusqu'au bout.

Je souhaite également exprimer ma reconnaissance envers mon directeur de recherche pour son accompagnement rigoureux, sa disponibilité et la qualité de ses orientations. Son expertise et son engagement ont été déterminants dans la réalisation de ce travail et ont renforcé mon intérêt pour l'intelligence artificielle et les modèles de langage. Au-delà de l'encadrement académique, vous m'avez transmis une exigence intellectuelle qui guidera ma future carrière.

Mes remerciements vont également à mes amis de Trois-Rivières, Saliou Diop, Cheikh Fall et Muhammad Omar Peghoumma, pour leur soutien et leur présence durant cette période. Votre fraternité a transformé l'exil en une expérience humaine enrichissante.

Je tiens à saluer l'ensemble des professeurs du programme de maîtrise en mathématiques et informatique appliquées de l'UQTR pour la qualité de l'enseignement dispensé, ainsi que le personnel administratif et technique pour leur disponibilité et leur efficacité.

À tous ceux que je n'ai pu nommer individuellement, mais qui ont contribué à l'aboutissement de ce travail : recevez l'expression de ma sincère gratitude.

Table des matières

Table des figures

Liste des tableaux

Liste des abréviations

A100	NVIDIA A100-SXM4-80GB (modèle de GPU)
API	*Application Programming Interface* (Interface de Programmation)
BIM	*Building Information Modeling* (Modélisation des Informations du Bâtiment)
BLE	*Bluetooth Low Energy* (Bluetooth Basse Consommation)
FP16	*Float Point 16 bits* (Virgule Flottante 16 bits)
GPS	*Global Positioning System* (Système de Positionnement Global)
GPU	*Graphics Processing Unit* (Unité de Traitement Graphique)
IA	Intelligence Artificielle
IoT	*Internet of Things* (Internet des Objets)
JSON	*JavaScript Object Notation*
JSONL	*JSON Lines* (JSON en lignes)
LLM	*Large Language Model* (Grand Modèle de Langage)
LoRA	*Low-Rank Adaptation* (Adaptation de Rang Faible)
NaviUQTR	Système de Navigation UQTR
NLP	*Natural Language Processing* (Traitement du Langage Naturel)
PEFT	*Parameter-Efficient Fine-Tuning* (Fine-tuning Efficace en Paramètres)
PMR	Personne à Mobilité Réduite
RFID	*Radio Frequency Identification* (Identification par Radiofréquence)
SCP	*Cyber Physical Systems* (Systèmes Cyber-physiques)
UQTR	Université du Québec à Trois-Rivières
UWB	*Ultra-Wideband* (Ultra Large Bande)
V1	Version 1 (phase exploratoire)
V2	Version 2 (phase d'optimisation)
V2.1	Version 2.1 (phase finale)
VLN	*Vision-and-Language Navigation* (Navigation Vision-Langage)
WiFi	*Wireless Fidelity* (Fidélité Sans Fil)

Chapitre 1 – Introduction générale

1.1 Contexte général

Les bâtiments intelligents connaissent aujourd'hui une transformation profonde, portée par les avancées rapides des technologies de l'information, de l'Internet des objets (IdO, ou IoT de l'anglais *Internet of Things*, désignant l'interconnexion via Internet d'objets physiques embarquant des capteurs et des logiciels) et de l'intelligence artificielle (IA, ensemble des techniques permettant à des systèmes informatiques d'accomplir des tâches nécessitant normalement l'intelligence humaine). Grâce à l'intégration progressive de capteurs, de systèmes automatisés et d'outils de gestion énergétique avancés, ces infrastructures autrefois perçues comme statiques évoluent désormais vers des espaces dynamiques, capables de percevoir leur environnement et de s'adapter aux besoins de leurs occupants.

Parallèlement, les récents progrès réalisés en intelligence artificielle, et plus particulièrement le développement des grands modèles de langage (*Large Language Models* – LLMs), systèmes d'intelligence artificielle entraînés sur de vastes corpus textuels et capables de comprendre et de générer du langage naturel, ouvrent de nouvelles possibilités. Ces modèles permettent aujourd'hui de mieux comprendre le langage naturel, de raisonner sur des actions complexes et de générer des instructions adaptées à divers contextes. Ils constituent ainsi une opportunité pour concevoir des formes d'interactions inédites entre les utilisateurs et les espaces bâtis.

Note sur la modélisation schématique

Le système NaviUQTR développé dans le cadre de ce mémoire repose sur une modélisation schématique et simplifiée d'un environnement intérieur universitaire inspiré du pavillon Ringuet de l'UQTR. Bien qu'inspirée d'un bâtiment réel, la représentation topologique, les codes de salles et la disposition des espaces ont été volontairement simplifiés et adaptés à des fins de démonstration et de validation méthodologique.

Cette approche schématique présente plusieurs justifications scientifiques. D'abord, elle permet de concentrer l'évaluation sur les mécanismes fondamentaux du système : interprétation du langage naturel, planification d'itinéraires, génération d'instructions contextualisées, sans les complications liées à la complexité architecturale exhaustive d'un bâtiment réel. Ensuite, elle facilite la reproduction et l'extension de ce travail par d'autres chercheurs, la méthodologie développée étant indépendante de toute configu-

ration particulière. Enfin, elle constitue une preuve de concept dont les principes sont directement transposables à la configuration réelle du pavillon Ringuet ou à tout autre bâtiment moyennant une phase de modélisation précise avec les plans architecturaux authentiques.

Les choix méthodologiques spécifiques relatifs à cette modélisation schématique seront détaillés au chapitre 3, dans la section dédiée à la représentation topologique du bâtiment.

1.2 Problématique

La mobilité en milieu intérieur demeure un défi majeur dans les grandes infrastructures, telles que les universités, les hôpitaux, les administrations publiques ou les complexes industriels. Bien que la navigation en plein air ait longtemps bénéficié de solutions solides basées sur le GPS (*Global Positioning System*), l'environnement intérieur ne dispose pas encore de technologies standardisées permettant une localisation et une orientation fiables.

Les approches existantes présentent plusieurs limites. Beaucoup s'appuient sur des algorithmes rigides, peu adaptatifs face à la diversité des situations rencontrées dans un bâtiment réel. Elles sont incapables d'interpréter le langage naturel des usagers, exigent parfois des équipements spécialisés ou coûteux, et s'adaptent mal aux reconfigurations fonctionnelles ou structurelles. De plus, leur interaction avec l'utilisateur reste souvent peu personnalisée.

Bien que les récents développements en IA, notamment les LLMs, aient démontré des capacités remarquables en compréhension du langage et en raisonnement procédural, leur application concrète à la navigation intérieure demeure limitée. Ce constat conduit à la question centrale de ce mémoire :

Comment concevoir un bâtiment intelligent capable d'interpréter les requêtes formulées par ses utilisateurs et de générer des itinéraires précis, contextualisés et adaptés, en exploitant le potentiel des modèles de langage ?

1.3 Objectif général

Ce travail vise à concevoir et à valider une approche de navigation intérieure permettant au bâtiment, à travers un système de guidage intelligent, d'assurer trois fonctions principales. D'abord, interpréter les requêtes en langage naturel, c'est-à-dire comprendre les demandes telles qu'elles sont formulées spontanément, sans contraintes

terminologiques spécifiques. Ensuite, générer automatiquement des itinéraires optimaux qui tiennent compte de la topologie du bâtiment, des contraintes d'accès et, le cas échéant, des préférences de l'utilisateur. Enfin, produire des instructions de navigation claires, contextualisées et personnalisées, facilement compréhensibles par différents profils d'usagers.

L'approche proposée s'appuie sur l'intégration et l'adaptation d'un modèle de langage spécialisé, combinant compréhension linguistique et raisonnement spatial, pour fournir une assistance en navigation précise et intuitive.

1.4 Objectifs spécifiques

Pour atteindre cet objectif général, le travail a été structuré autour des objectifs spécifiques suivants :

1. Modéliser la configuration interne du bâtiment sous la forme d'un graphe spatial représentant sa topologie et ses principaux points d'intérêt.
2. Constituer un corpus de trajectoires synthétiques diversifié et équilibré, généré de manière contrôlée à partir des destinations disponibles dans le modèle de bâtiment, afin de soutenir l'entraînement et l'évaluation du système.
3. Pré-entraîner un modèle de langage pour lui transmettre une compréhension élémentaire des relations spatiales et des principes de navigation.
4. Procéder à un ajustement fin du modèle sur les données propres au bâtiment étudié (UQTR) afin de renforcer sa capacité à raisonner dans ce contexte particulier.
5. Développer un pipeline complet capable d'interpréter les requêtes, de générer les chemins appropriés et de produire des instructions adaptées aux usagers.
6. Comparer les performances du système proposé aux approches existantes, notamment PathGen-LLM [1], MapGPT [2] et Zhang et al. [3], en termes de précision d'identification de destination et de pertinence des instructions générées.
7. Évaluer la qualité des trajectoires produites ainsi que la clarté, l'efficacité et la pertinence des instructions générées.

1.5 Pertinence scientifique et originalité de la démarche

Les recherches actuelles sur les bâtiments intelligents se concentrent majoritairement sur la gestion énergétique, la sécurité ou l'optimisation des infrastructures techniques.

En revanche, la dimension communicationnelle, c'est-à-dire la capacité du bâtiment à interagir directement avec ses occupants, demeure encore peu explorée.

L'essor des grands modèles de langage permet cependant d'envisager des bâtiments capables non seulement de percevoir leur environnement, mais aussi de comprendre les demandes des usagers et d'y répondre de manière contextualisée.

La démarche proposée se distingue par cette orientation centrée sur l'humain. Plutôt que d'utiliser les LLMs pour des tâches purement analytiques, ce travail les mobilise pour interpréter des requêtes en langage naturel et fournir des indications de navigation adaptées à un environnement réel. Il s'agit d'une démarche appliquée et concrète, qui vise à rapprocher les avancées théoriques en IA des besoins pratiques des usagers.

1.6 Enjeux méthodologiques et défis de conception

Ce projet soulève plusieurs défis méthodologiques. D'abord, le langage humain est naturellement ambigu : une requête telle que « Comment aller au labo ? » nécessite de déduire la destination précise, la position de l'usager et la forme d'instruction la plus appropriée.

Ensuite, la traduction de l'espace physique en une représentation exploitable par un modèle de langage constitue une difficulté majeure. Il ne s'agit pas seulement de cartographier des salles et des couloirs, mais de structurer l'information de manière à permettre un raisonnement spatial cohérent.

Enfin, la qualité des instructions générées doit être appréciée sous plusieurs angles : l'exactitude des itinéraires, la clarté des formulations, l'adaptation au profil de l'utilisateur et la capacité à rendre la navigation fluide.

Ces enjeux justifient la démarche progressive adoptée, articulant une modélisation spatiale rigoureuse, l'adaptation linguistique du modèle et une évaluation quantitative et qualitative.

1.7 Structure du mémoire

Ce mémoire est organisé comme suit. Le **chapitre 1** présente l'introduction générale, le contexte, la problématique et les objectifs du travail. Le **chapitre 2** est consacré à une revue de littérature sur les bâtiments intelligents, la navigation intérieure et les modèles de langage. Le **chapitre 3** décrit la méthodologie et le pipeline proposés. Le **chapitre 4** présente les résultats expérimentaux et en propose une analyse. Le

chapitre 5 offre une conclusion générale, discute les limites de l'étude et ouvre sur des perspectives de recherche.

1.8 Conclusion

Cette introduction a posé les jalons du travail présenté dans ce mémoire. Le contexte évolutif des bâtiments intelligents a été présenté, les défis persistants de la navigation intérieure ont été mis en lumière et une question de recherche visant à combiner raisonnement spatial et traitement du langage naturel a été formulée.

Les objectifs principaux ainsi que les choix méthodologiques qui structurent la démarche ont également été exposés, en insistant sur l'ancrage dans un environnement réel et sur la volonté de replacer l'humain au centre des interactions avec le bâtiment.

Le chapitre suivant sera consacré à une revue de littérature approfondie, qui établira les fondements théoriques et techniques nécessaires et situera la contribution de ce travail dans le paysage scientifique actuel.

Chapitre 2 – Revue de littérature

2.1 Introduction

La littérature scientifique consacrée aux bâtiments intelligents, à la navigation intérieure et à l'IA a connu une évolution remarquable au cours de la dernière décennie. D'une part, les progrès réalisés en matière d'IoT, de systèmes cyber-physiques (SCP, systèmes intégrant étroitement des composants informatiques et des processus physiques au moyen de capteurs et d'actionneurs [4]) et d'intelligence ambiante ont progressivement transformé les bâtiments en environnements connectés, capables d'interagir avec leurs occupants. D'autre part, l'essor de l'apprentissage profond et, plus récemment, l'émergence des grands modèles de langage ont ouvert des horizons nouveaux pour la compréhension du langage naturel, la modélisation spatiale et la génération d'itinéraires personnalisés.

Ce chapitre vise à exposer les fondements théoriques et technologiques sur lesquels repose le travail de recherche présenté dans ce mémoire. Les développements qui suivent s'intéressent d'abord à l'évolution des bâtiments intelligents et à leurs architectures SCP, avant d'examiner comment l'espace intérieur peut être modélisé de manière exploitable pour la navigation. Les approches classiques de navigation intérieure seront ensuite analysées, puis les avancées récentes issues du domaine *vision-langage* et des agents incarnés. Enfin, une attention particulière sera accordée aux LLMs et à leur potentiel pour le raisonnement spatial et la génération d'itinéraires. Cette exploration permettra de mettre en évidence les lacunes actuelles et de justifier l'approche proposée dans ce mémoire.

2.2 Bâtiments intelligents et cyber-physiques

2.2.1 Évolution historique et transformation des bâtiments

La notion de bâtiment intelligent est apparue dans les années 1980, à une époque où elle se limitait essentiellement à l'automatisation des infrastructures techniques. Avec l'avènement des réseaux de capteurs, de l'IoT et de l'intelligence artificielle, les bâtiments ont progressivement évolué vers des espaces véritablement interactifs et adaptatifs [5], [6]. Cette transformation se traduit aujourd'hui par l'émergence d'environnements capables de percevoir leur contexte, d'analyser les données captées, de prendre des décisions et d'interagir de manière pertinente avec leurs usagers.

2.2.2 IoT, *edge computing* et systèmes cyber-physiques

Les bâtiments intelligents modernes s'appuient sur des SCP complexes qui intègrent réseaux de capteurs, l'informatique en périphérie (*edge computing*), paradigme de traitement des données au plus près des sources de collecte, et l'informatique en nuage (*cloud computing*) et algorithmes d'analyse avancés. Ces technologies permettent une supervision en temps réel, facilitent la maintenance prédictive, optimisent la consommation énergétique et contribuent à une meilleure compréhension des comportements d'occupation [7]. Malgré ces avancées considérables, l'assistance directe aux usagers, en particulier pour la navigation intérieure, demeure un domaine encore peu développé. C'est précisément cette lacune qui motive en grande partie le travail présenté ici.

Dans le contexte des bâtiments intelligents, les SCP se manifestent par la collecte en temps réel de données environnementales (température, occupation, consommation énergétique), leur traitement par des algorithmes embarqués (*edge computing*) ou déportés (*cloud computing*), et l'activation de systèmes de contrôle physiques en réponse. La navigation intérieure constitue une application naturelle de ce paradigme : les requêtes des usagers représentent des entrées symboliques qui déclenchent des traitements informatiques (interprétation linguistique, calcul d'itinéraire) dont les sorties influencent directement les déplacements physiques dans l'environnement bâti. NaviUQTR s'inscrit dans cette vision en proposant une couche d'interaction linguistique au-dessus d'une représentation topologique du bâtiment, contribuant ainsi à la dimension communicante des SCP.

2.2.3 Intelligence ambiante et interaction humain–bâtiment

L'intelligence ambiante vise à rendre les espaces sensibles au contexte et aux besoins des utilisateurs. Les recherches récentes soulignent l'importance de centrer davantage les systèmes sur l'humain, notamment à travers des interactions en langage naturel [8]. Or, la navigation intérieure reste un aspect encore peu exploré, alors qu'elle constitue un besoin essentiel dans les grands campus universitaires, les hôpitaux et les bâtiments administratifs. Cette observation renforce la pertinence de l'approche proposée, qui cherche justement à combler ce vide en proposant un système capable de comprendre et de répondre aux demandes de navigation formulées en langage courant.

2.3 Modélisation de l'espace intérieur

2.3.1 Modèles géométriques

Les représentations géométriques, qu'il s'agisse de plans en deux dimensions, de modèles *Building Information Modeling* (BIM) ou de surfaces tridimensionnelles, constituent les premières approches de modélisation des bâtiments. Bien qu'elles offrent un niveau de détail élevé, ces représentations sont difficiles à exploiter directement pour la navigation [9]. Elles manquent notamment d'informations sémantiques, et leur granularité continue complique la planification d'itinéraires. Pour ces raisons, une approche topologique a été privilégiée dans ce travail, approche qui semble mieux adaptée aux contraintes de la navigation intérieure.

2.3.2 Modèles topologiques et graphes spatiaux

La modélisation topologique sous forme de graphes constitue aujourd'hui le paradigme dominant pour la représentation des espaces intérieurs.Dans cette approche, les lieux sont représentés par des nœuds (salles, intersections, points de repère), tandis que les arêtes symbolisent les connexions possibles entre ces espaces, telles que les couloirs ou les portes [10], [11]. Cette abstraction offre un cadre particulièrement efficace pour la planification d'itinéraires. De plus, ces graphes peuvent être enrichis d'attributs variés : distances métriques, contraintes d'accessibilité, coûts de déplacement, ou encore informations contextuelles [12]. Cette flexibilité a conduit à adopter cette représentation comme fondement du système proposé.

2.3.3 Modèles hiérarchiques multi-niveaux

Pour les bâtiments comportant plusieurs étages, les graphes multi-niveaux permettent de représenter de manière explicite les connexions verticales, telles que les escaliers, les ascenseurs et les rampes d'accès, améliorant ainsi la précision des trajets dans des infrastructures complexes. Le standard *IndoorGML* fournit un cadre formel pour décrire des espaces intérieurs sous forme de cellules connectées [13]. Bien que ce standard n'ait pas été adopté dans son intégralité dans le cadre de ce travail, les principes qu'il propose ont influencé la manière de structurer le graphe spatial du bâtiment étudié.

2.3.4 Extraction automatique de graphes

Des travaux récents ont exploré la combinaison de techniques de vision par ordinateur et d'apprentissage profond pour extraire automatiquement des graphes à partir de plans d'architecture ou d'images de cartes [14]. Ces méthodes réduisent consi-

dérablement la charge de modélisation manuelle, mais elles requièrent des données visuelles de qualité et demeurent sensibles aux variations de format. Dans le cadre de cette étude, une approche semi-automatique a été privilégiée, combinant extraction algorithmique et vérification manuelle, afin de garantir la fiabilité du graphe obtenu.

2.3.5 Contraintes d'accessibilité

Les systèmes modernes de navigation doivent impérativement intégrer des contraintes d'accessibilité, telles que la prise en compte des personnes à mobilité réduite, l'existence d'escaliers ou d'ascenseurs, et la présence de zones interdites ou temporairement inaccessibles. Les graphes enrichis sémantiquement permettent une planification plus réaliste et véritablement personnalisée [12]. Cette dimension a semblé essentielle, car elle conditionne directement l'utilité pratique du système proposé.

2.4 Navigation intérieure : méthodes classiques

2.4.1 Technologies de localisation intérieure

Plusieurs technologies ont été développées au fil des années pour localiser un usager à l'intérieur d'un bâtiment. La reconnaissance d'empreintes WiFi (*WiFi fingerprinting*), bien que répandu, demeure sensible aux variations environnementales et nécessite une phase de calibration fastidieuse [15]. Le *Bluetooth Low Energy* (BLE) offre une solution peu coûteuse, mais sa précision reste variable selon la densité des balises et les interférences [16]. La technologie à bande ultralarge (*Ultra-wideband*, UWB) se distingue par son exactitude remarquable, mais son déploiement engendre des coûts élevés qui limitent son adoption à grande échelle [17]. La technologie RFID est principalement utilisée pour la localisation d'objets plutôt que de personnes [18]. Enfin, la vision par ordinateur, bien que performante dans certains contextes, s'avère peu adaptée à un usage direct par les usagers humains [14].

Chacune de ces technologies présente des limites qui compliquent le développement d'un système de navigation robuste et universel. C'est pourquoi, dans l'approche développée ici, le choix a été fait de ne pas dépendre d'une technologie de localisation particulière, mais plutôt de s'appuyer sur une représentation abstraite du bâtiment, exploitable indépendamment du système de positionnement utilisé.

2.4.2 Algorithmes de recherche de chemin

Les algorithmes classiques de recherche de chemin, tels que l'algorithme de Dijkstra [19] ou A* [20], sont largement utilisés pour la navigation intérieure. Des variantes

multi-critères permettent d'intégrer des contraintes supplémentaires, comme l'accessibilité ou la minimisation du temps de trajet [21]. Ces algorithmes, bien qu'efficaces, ne prennent généralement pas en compte la compréhension du langage naturel ni la personnalisation des instructions. Le travail présenté vise précisément à combler cette lacune en associant ces méthodes algorithmiques éprouvées à un modèle de langage capable d'interpréter les demandes des usagers et de générer des consignes adaptées.

2.4.3 Systèmes de guidage traditionnels

Les systèmes de guidage existants, qu'il s'agisse d'instructions textuelles statiques, de cartes interactives ou d'applications mobiles, souffrent généralement d'un manque de personnalisation, d'une faible capacité d'adaptation et d'une interaction limitée avec l'usager [22]. Ces systèmes supposent souvent que l'utilisateur connaît déjà la terminologie propre au bâtiment ou qu'il est capable de lire une carte complexe. L'approche proposée cherche à dépasser ces limites en permettant une interaction naturelle, où l'usager peut simplement formuler sa destination en langage courant.

2.5 *Vision-and-Language Navigation* (VLN)

2.5.1 Émergence du domaine

Le domaine de la *Vision-and-Language Navigation* (VLN) est apparu avec la tâche *Room-to-Room* (R2R) introduite par Anderson et al. [23]. Il s'agit d'une tâche multimodale où un agent virtuel doit suivre des instructions textuelles dans un environnement photoréaliste simulé. Cette approche a ouvert de nouvelles perspectives en démontrant qu'il était possible de combiner compréhension linguistique et perception visuelle pour guider un agent dans un espace complexe.

2.5.2 Architectures VLN

Les architectures développées dans le cadre du VLN combinent généralement des encodeurs linguistiques, des encodeurs visuels et des contrôleurs séquentiels [24], [25]. Les modèles récents s'appuient sur des *transformers* multimodaux, capables d'aligner de manière précise les représentations textuelles et visuelles. Ces avancées témoignent de la puissance des approches d'apprentissage profond pour la navigation guidée par le langage.

2.5.3 Environnements simulés

Il convient toutefois de noter que les environnements utilisés en VLN sont presque exclusivement simulés, tels que *Matterport3D* ou *Habitat*. Leur transférabilité vers des bâtiments réels reste limitée, en raison des différences entre les espaces virtuels et les contraintes concrètes des environnements physiques [26]. Cette observation a conduit à privilégier une approche fondée sur une représentation abstraite du bâtiment, plus facilement adaptable à des contextes réels.

2.5.4 Limites des modèles VLN

Les modèles VLN supposent généralement la présence d'un agent robotique équipé de caméras et capable de se déplacer physiquement dans l'environnement. Ils exigent un flux visuel continu et ne sont pas conçus pour des contextes où l'utilisateur est un être humain naviguant de manière autonome, comme c'est le cas dans un campus universitaire ou un hôpital.Cette inadéquation entre les hypothèses des modèles VLN et les besoins pratiques de la navigation humaine justifie l'approche alternative développée dans ce mémoire.

2.6 Agents incarnés et modèles cognitifs

2.6.1 Agents séquentiels

Les agents basés sur l'apprentissage par renforcement, tels que ceux développés avec les algorithmes *Asynchronous Advantage Actor-Critic* (A3C) [27] ou *Proximal Policy Optimization* (PPO) [28], ont démontré leur capacité à apprendre des séquences d'actions complexes dans des environnements tridimensionnels. Toutefois, leur généralisation à des environnements non vus durant l'entraînement demeure un défi majeur [29]. Ces agents, bien qu'impressionnants dans des contextes contrôlés, peinent à s'adapter à la variabilité des bâtiments réels. Cette limite renforce l'intérêt d'une approche fondée sur un modèle de langage, qui peut potentiellement généraliser plus facilement grâce à sa compréhension symbolique de l'espace.

2.6.2 Agents multimodaux

Des environnements comme *BabyAI* [30], *ALFRED* [31] ou *BEHAVIOR* [32] ont démontré la capacité des agents à comprendre des instructions complexes et à effectuer des tâches sophistiquées dans des environnements simulés. Ces travaux illustrent la convergence entre compréhension linguistique et raisonnement spatial.

La Figure 2.1 présente une architecture multi-agents orchestrée développée par Kalyuzhnaya et al. pour la gestion des villes intelligentes [33]. Cette approche modulaire repose sur un agent conducteur central (*agent-conductor*) qui coordonne plusieurs agents secondaires spécialisés. Un premier agent gère l'appel d'outils externes via des interfaces de programmation applicative (API, *Application Programming Interface*), tandis qu'un second agent effectue des opérations de recherche augmentée par génération (*RAG operation*) en interrogeant une base vectorielle de documents réglementaires. Un troisième agent synthétise les réponses finales, et des agents validateurs assurent la qualité des résultats avant leur transmission à l'utilisateur via une interface conversationnelle intégrée au système d'information de la ville.

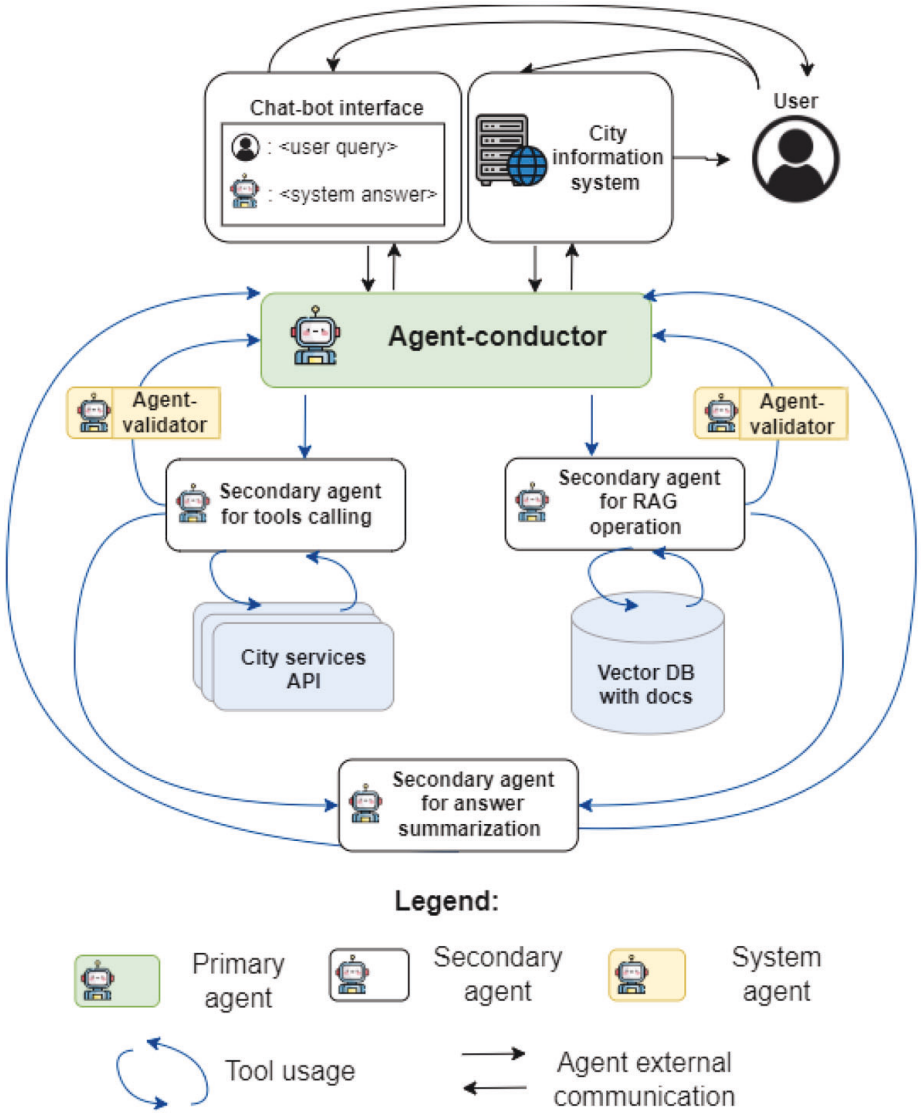

FIGURE 2.1 – Architecture multi-agents orchestrée pour un système d'information urbaine intelligente[33].

Cette architecture modulaire favorise la robustesse et la maintenabilité du système, tout en permettant une spécialisation fine de chaque composante. Les auteurs rap-

portent une précision de routage des requêtes de 94% à 99% selon les modèles LLM utilisés, démontrant l'efficacité de cette orchestration pour traiter des questions hétérogènes sur l'environnement urbain.

Néanmoins, ces approches restent principalement centrées sur des tâches de gestion urbaine et ne sont pas directement transposables à des contextes où l'usager humain doit être guidé verbalement dans un environnement intérieur.Le travail présenté s'inspire de ces avancées tout en les adaptant à un cadre où l'interaction se fait principalement par le biais d'instructions textuelles de navigation, sans nécessiter de perception visuelle en temps réel.

2.7 LLMs et raisonnement spatial

2.7.1 Capacités générales des LLMs

Les LLMs, tels que GPT-3 [34], PaLM [35], LLaMA [36] ou Mistral [37], ont démontré une capacité remarquable à générer du texte cohérent, à planifier des séquences d'actions et à raisonner sur des problèmes abstraits. Ces modèles, entraînés sur des corpus textuels massifs, ont acquis une compréhension implicite de nombreux domaines, y compris des aspects liés à la spatialité et à la navigation.

La Figure 2.2 illustre une architecture d'intégration de LLMs pré-entraînés dans un système de gestion de bâtiment intelligent multi-zones développée par Papaioannou et al.[38]. Le système collecte diverses données contextuelles (informations temporelles, conditions météorologiques, données de capteurs de température et d'humidité, production solaire photovoltaïque) ainsi que les données d'occupation pour alimenter un modèle de langage pré-entraîné.

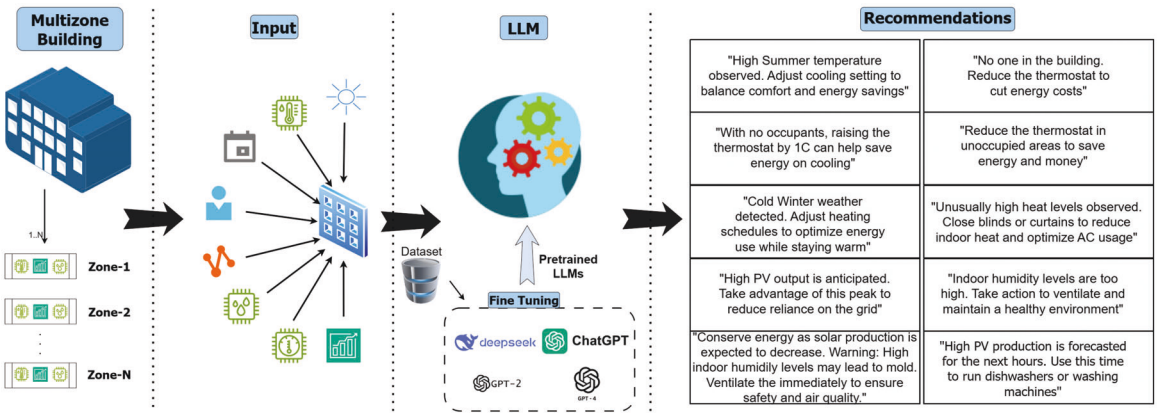

FIGURE 2.2 – Architecture d'intégration de LLMs pré-entraînés pour la génération de recommandations intelligentes dans un bâtiment multi-zones [38].

Après une phase d'affinage spécifique au domaine de la gestion énergétique des bâtiments, le modèle génère des recommandations personnalisées en langage naturel. Ces recommandations couvrent des aspects variés tels que l'ajustement des paramètres de chauffage et de climatisation, l'optimisation de l'utilisation des équipements en fonction de la production solaire, ou encore des alertes concernant la qualité de l'air intérieur. Cette approche démontre le potentiel des LLMs à interpréter des contextes complexes et à produire des consignes adaptées.

Au-delà de la génération de recommandations, les LLMs peuvent également opérer selon un paradigme agent autonome, comme l'illustre la Figure 2.3. Dans cette architecture proposée par Kalyuzhnaya et al. [33], un agent LLM central suit un cycle itératif d'instruction, d'action et de réflexion (*reflection*).

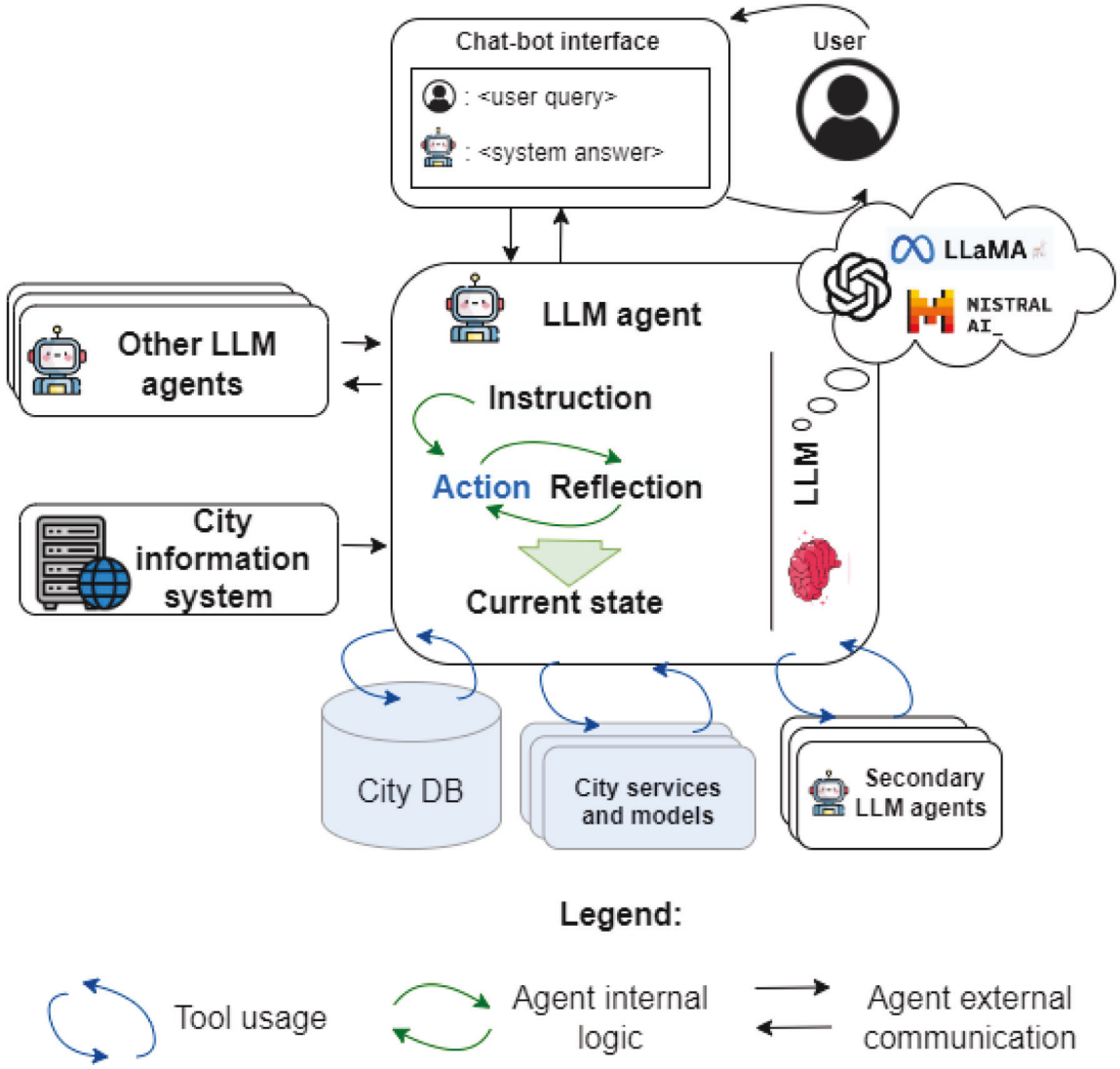

FIGURE 2.3 – Architecture d'un agent LLM autonome avec cycle de raisonnement Action-Reflection pour la gestion de villes intelligentes [33].

Ce mécanisme permet à l'agent d'ajuster dynamiquement son comportement en fonction de l'état actuel du système et des retours d'information. L'agent peut interroger des bases de données urbaines, invoquer des services externes via API, et même collaborer avec d'autres agents LLM secondaires pour accomplir des tâches complexes. Cette approche agentique contraste avec les systèmes purement réactifs en introduisant une capacité d'auto-évaluation et d'amélioration continue.

Cette capacité générale constitue un atout majeur pour l'approche proposée, car elle permet d'exploiter un savoir linguistique riche sans nécessiter une modélisation explicite de chaque aspect du problème. Toutefois, contrairement aux travaux de Papaioannou et al[38]. qui se concentrent sur la gestion énergétique, ou à ceux de Kalyuzhnaya et al[33]. axés sur la planification urbaine stratégique, l'objectif du présent travail est d'adapter ces méthodologies au domaine spécifique de la navigation intérieure dans les bâtiments.

2.7.2 Raisonnement spatial

Des études récentes montrent que les LLMs peuvent inférer des relations spatiales à partir de descriptions textuelles ou générer des instructions de navigation cohérentes [39], [40]. Bien que leur compréhension spatiale ne soit pas aussi précise que celle d'un système géométrique dédié, elle s'avère souvent suffisante pour des tâches de guidage de haut niveau. L'hypothèse formulée est qu'en combinant cette capacité de raisonnement spatial avec une représentation topologique explicite du bâtiment, il devient possible de générer des itinéraires à la fois précis et exprimés de manière naturelle.

2.7.3 Modèles spécialisés

Des modèles spécialisés ont récemment été développés pour exploiter les LLMs dans le contexte de la navigation. *PathGen-LLM* [1] explore la génération de trajectoires textuelles dans des environnements abstraits. *SpatialLM* [41] traite la reconstruction structurelle de scènes tridimensionnelles à partir de nuages de points, une tâche connexe mais distincte de la génération de chemins de navigation intérieure. *MapGPT* [2] se concentre sur la compréhension et la génération de descriptions d'environnements. Ces travaux pionniers démontrent la faisabilité d'une navigation fondée sur le langage, mais ils restent encore largement expérimentaux et peu adaptés aux bâtiments intérieurs complexes. Le travail présenté s'inscrit dans cette lignée, tout en cherchant à proposer un cadre méthodologique plus complet et applicable à des environnements réels.

2.7.4 Pré-entraînement et affinage

Les stratégies récentes combinent un pré-entraînement sur des corpus de trajectoires génériques et un affinage sur un environnement cible, ce qui améliore significativement les performances [1]. Cette approche en deux étapes permet au modèle d'acquérir d'abord une compréhension générale de la navigation, puis de s'adapter aux spécificités d'un bâtiment particulier. Cette stratégie a été adoptée dans le présent travail, en construisant un corpus de trajectoires synthétiques pour le pré-entraînement, puis en affinant le modèle sur un corpus synthétique généré à partir du modèle schématique du bâtiment de l'UQTR.

2.8 Synthèse critique et positionnement

L'analyse approfondie de la littérature met en évidence plusieurs lacunes fondamentales qui justifient la contribution du travail présenté. Premièrement, il n'existe pas, à ce jour, de système intégré combinant de manière cohérente une compréhension du langage naturel, une représentation topologique du bâtiment et une génération d'itinéraires personnalisés. Les approches existantes tendent à se concentrer sur l'un ou l'autre de ces aspects, sans proposer de cadre unifié.

Deuxièmement, les travaux en VLN, bien qu'impressionnants sur le plan technique, s'appuient sur des environnements simulés dont la transférabilité vers des bâtiments réels reste incertaine. Ces environnements supposent en outre la présence d'un agent robotique, ce qui ne correspond pas aux besoins des usagers humains naviguant de manière autonome.

Troisièmement, malgré les capacités prometteuses des grands modèles de langage en matière de raisonnement spatial, leur application directe à la navigation intérieure demeure rare et peu systématique. Les quelques travaux existants restent expérimentaux et n'ont pas encore donné lieu à des systèmes opérationnels.

Enfin, les systèmes de guidage traditionnels souffrent d'un manque flagrant de personnalisation. Ils ne prennent généralement pas en compte les préférences individuelles, les contraintes d'accessibilité ou le contexte particulier de chaque demande. Cette rigidité limite considérablement leur utilité pratique.

Le travail présenté cherche précisément à combler ces lacunes en proposant un système qui articule de manière cohérente une représentation topologique du bâtiment, un modèle de langage spécialisé et un mécanisme capable de transformer une requête utilisateur en une séquence d'instructions de navigation claires, contextualisées et

adaptées à un environnement réel. Cette approche intégrée constitue une contribution originale au domaine de la navigation intérieure intelligente.

2.9 Conclusion

Ce chapitre a présenté un panorama approfondi des travaux portant sur les bâtiments intelligents, la modélisation de l'espace intérieur, les technologies de navigation, les approches *vision-langage* et les grands modèles de langage. Cette exploration a permis de situer le travail présenté dans un contexte scientifique riche et en pleine évolution.

L'analyse de la littérature montre que, malgré des progrès importants, plusieurs limites persistent lorsque l'on cherche à concevoir un système de navigation intérieure réellement centré sur l'usager. Les approches traditionnelles de localisation et de guidage demeurent fortement dépendantes d'infrastructures coûteuses, manquent de flexibilité et ne permettent pas une interaction naturelle en langage courant. Les systèmes de modélisation spatiale ont certes évolué vers des représentations topologiques robustes, mais leur intégration avec des mécanismes d'interprétation linguistique reste limitée. Les travaux en *Vision-and-Language Navigation* et sur les agents incarnés démontrent la capacité des modèles modernes à raisonner sur des environnements complexes, mais leurs applications se situent principalement dans des environnements simulés, souvent éloignés des contraintes des bâtiments réels.

Parallèlement, l'émergence des grands modèles de langage ouvre des perspectives nouvelles pour la compréhension sémantique, la planification textuelle et le raisonnement spatial symbolique. Des travaux récents, tels que *PathGen-LLM*, *SpatialLM* ou *MapGPT*, montrent qu'il est désormais possible d'exploiter un modèle de langage pour interpréter des requêtes de navigation et générer des itinéraires. Toutefois, la plupart de ces méthodes demeurent expérimentales, peu adaptées aux bâtiments intérieurs complexes, et surtout dépourvues d'un cadre méthodologique unifié permettant leur application concrète. L'ensemble de cette revue fait émerger un espace clair de contribution : la nécessité d'un système intégré qui combine, dans un même *pipeline*, une représentation topologique du bâtiment, un modèle de langage spécialisé et un mécanisme capable de transformer une requête utilisateur en une séquence d'instructions de navigation cohérentes, contextualisées et adaptées à un environnement réel. Le prochain chapitre est consacré à la méthodologie développée dans le cadre de ce mémoire. Y seront présentées la structuration du bâtiment étudié sous forme de graphe, la conception du corpus de trajectoires, les étapes de pré-entraînement et d'affinage du modèle de langage, ainsi que l'architecture du *pipeline* de navigation. Cette méthodologie constitue l'ensemble des fondations nécessaires pour répondre à la question de recherche formulée dans l'introduction générale.

Chapitre 3 – Méthodologie

3.1 Introduction

Ce chapitre présente la méthodologie développée pour concevoir un système de navigation intérieure intelligent capable d'interpréter des requêtes en langage naturel et de générer des itinéraires personnalisés. L'approche proposée repose sur quatre piliers fondamentaux : la modélisation spatiale d'un environnement intérieur sous forme de graphe topologique, la constitution d'un corpus d'entraînement pour l'adaptation du modèle de langage, l'affinage itératif d'un modèle Mistral-7B avec la technique par adaptation par matrices de rang faible (LoRA, *Low-Rank Adaptation*), et la mise en œuvre d'un pipeline complet intégrant compréhension linguistique et planification d'itinéraires.

La présentation commence par une vue d'ensemble de l'architecture du système, puis détaille chacune de ses composantes. La première section explique comment l'environnement a été modélisé sous forme de graphe multi-niveaux, en tenant compte de sa topologie, de ses points d'intérêt et de ses contraintes d'accessibilité. La section suivante aborde la constitution du corpus d'entraînement, élément central qui a évolué à travers trois versions successives. Le processus d'adaptation du modèle de langage est ensuite présenté, en détaillant la stratégie d'affinage itératif qui a permis d'améliorer progressivement les performances du système de 37,5% à 93,8% de précision. Enfin, le pipeline complet de navigation est décrit, de l'interprétation des requêtes à la génération d'instructions, avant de conclure par la présentation des métriques d'évaluation et de l'environnement technique.

3.2 Vue d'ensemble de l'architecture

Le système NaviUQTR repose sur une architecture modulaire en pipeline qui transforme progressivement une requête en langage naturel en un itinéraire détaillé et contextualisé. Cette approche permet de séparer les préoccupations et facilite l'évolutivité du système.

La Figure 3.1 présente l'organisation générale du système. Le traitement s'effectue en cinq étapes principales qui collaborent de manière séquentielle pour produire des instructions de navigation naturelles et précises. Chaque module assume une respon-

sabilité spécifique dans le pipeline de traitement, depuis l'analyse linguistique jusqu'à la génération de texte optimisée.

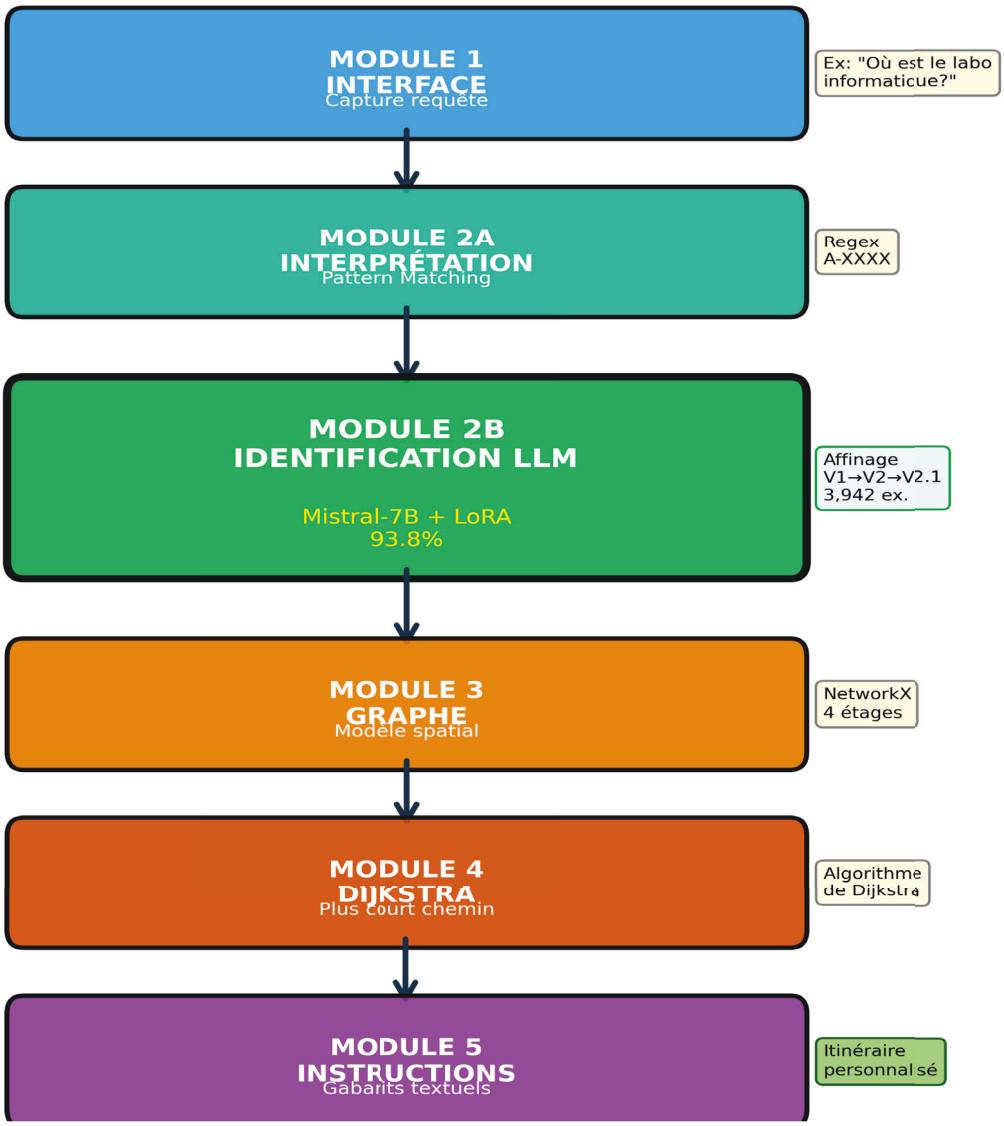

FIGURE 3.1 – Architecture générale du système NaviUQTR.

Le traitement débute lorsque l'utilisateur formule une requête en langage naturel, telle que « Où se trouve la salle A-2110 ? » ou « Comment aller au laboratoire de recherche ? ». Le premier module (Module 1) capture cette requête via l'interface utilisateur. Le deuxième module (Module 2A) analyse ensuite la requête par correspondance de motifs (*pattern matching*) pour détecter la présence éventuelle de codes de salles explicites ou de termes fonctionnels reconnus. Le troisième module (Module 2B), qui constitue le cœur innovant du système, fait appel au modèle Mistral-7B affiné avec LoRA pour identifier précisément la destination souhaitée, même lorsque la requête utilise des formulations variées ou des alias sémantiques. Une fois la destination déterminée, le

quatrième module (Module 3) exploite le graphe topologique du bâtiment, puis le cinquième module (Module 4) calcule le chemin optimal via l'algorithme de Dijkstra, en tenant compte des contraintes de déplacement le long des couloirs et des transitions verticales par le noyau central. Enfin, le sixième module (Module 5) transforme ce chemin abstrait en instructions textuelles structurées, suivant des gabarits prédéfinis qui garantissent la cohérence et la complétude des consignes.

Cette architecture modulaire présente plusieurs avantages méthodologiques. D'abord, la séparation nette des responsabilités facilite le développement et le débogage de chaque composant de manière indépendante. Ensuite, elle permet d'envisager des améliorations ciblées, par exemple en remplaçant le module d'affinage par un modèle plus performant sans affecter les autres étapes du traitement. De plus, cette organisation reflète les bonnes pratiques en génie logiciel, notamment le principe de responsabilité unique et la séparation des préoccupations. Enfin, elle s'inscrit dans une vision évolutive du système où de nouveaux modules pourraient être intégrés, tels qu'un gestionnaire de préférences utilisateur ou un module de personnalisation contextuelle.

Un aspect méthodologique important de cette architecture réside dans sa constance à travers les différentes versions expérimentales développées dans ce travail. Comme l'illustre la Figure 3.2, seul le Module 2B (identification par LLM) évolue entre les versions V1, V2 et V2.1, les autres composants demeurant identiques. Cette approche permet d'isoler l'impact de la stratégie d'affinage sur les performances globales du système, renforçant ainsi la validité scientifique de l'évaluation comparative.

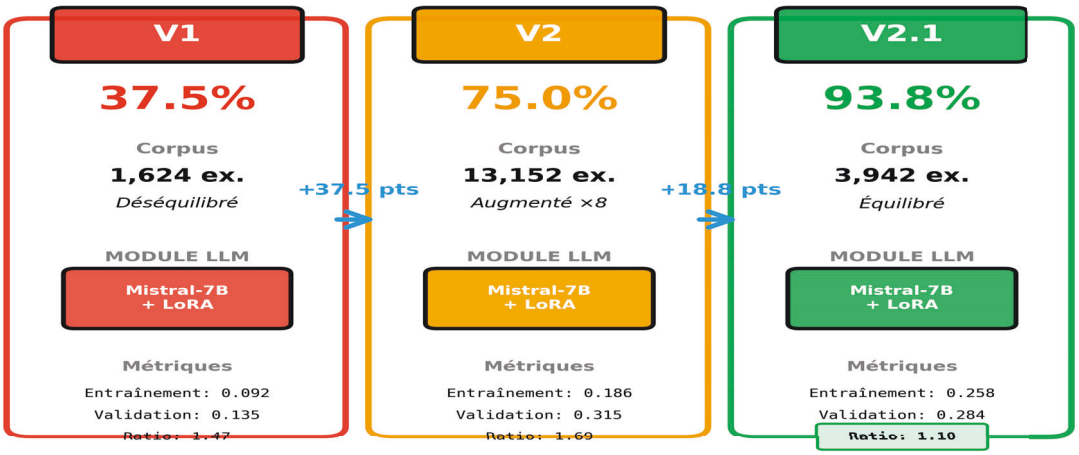

FIGURE 3.2 – Évolution du module LLM à travers les versions **V1**, **V2** et **V2.1**. L'architecture globale reste constante ; seuls les poids du module 2B (LLM affiné) changent selon la stratégie de composition du corpus d'entraînement adoptée à chaque itération (taille, équilibre entre catégories de requêtes et proportion d'alias sémantiques).

Les sections suivantes détaillent l'implémentation de chaque composant, en commençant par la modélisation spatiale du bâtiment qui alimente les modules de recherche de chemin (section 3.3), puis la constitution du corpus d'entraînement (section 3.4), et enfin l'adaptation du modèle de langage par affinage itératif (section 3.5).

3.3 Modélisation spatiale du bâtiment

La modélisation spatiale constitue le socle du système de navigation. Une représentation schématique du pavillon Ringuet comprenant quatre étages interconnectés a été développée, puis formalisée sous forme de graphe pour permettre le calcul de chemins optimaux.

3.3.1 Vue d'ensemble de la structure

Le pavillon Ringuet de l'UQTR est un bâtiment de quatre étages accueillant des salles de classe, des laboratoires et des espaces de réunion. Pour valider notre système NaviUQTR, nous avons modélisé de manière schématique ce bâtiment en représentant les éléments essentiels à la navigation intérieure : couloirs, salles, escaliers, ascenseurs et passerelles.

Configuration générale du bâtiment

Chaque étage est structuré autour d'un réseau de couloirs organisé en grille 3×3, formant neuf intersections principales. Au centre de cette grille se trouve le noyau du bâtiment, comprenant les escaliers et l'ascenseur, permettant la circulation verticale entre les étages. Cinq salles sont réparties sur chaque étage, leurs portes donnant directement sur les couloirs. Cette modélisation simplifiée conserve les caractéristiques topologiques essentielles du bâtiment réel tout en facilitant la représentation graphique et le calcul d'itinéraires.

Le rez-de-chaussée (1er étage) présente une particularité : il constitue le point d'entrée principal du bâtiment et accueille la passerelle Albert-Tessier, reliant le pavillon Ringuet au pavillon voisin. Cette passerelle est intégrée au graphe de navigation comme un nœud spécial, permettant au système de calculer des itinéraires incluant le déplacement entre pavillons.

Représentation schématique par étage

Les figures suivantes 3.3, 3.4, 3.5 et 3.6 présentent la disposition détaillée de chaque étage du pavillon Ringuet. Chaque plan illustre le réseau de couloirs (en bleu clair),

les salles (rectangles blancs avec codes d'identification), le noyau central (escaliers et ascenseur), ainsi que les éléments spécifiques à chaque niveau.

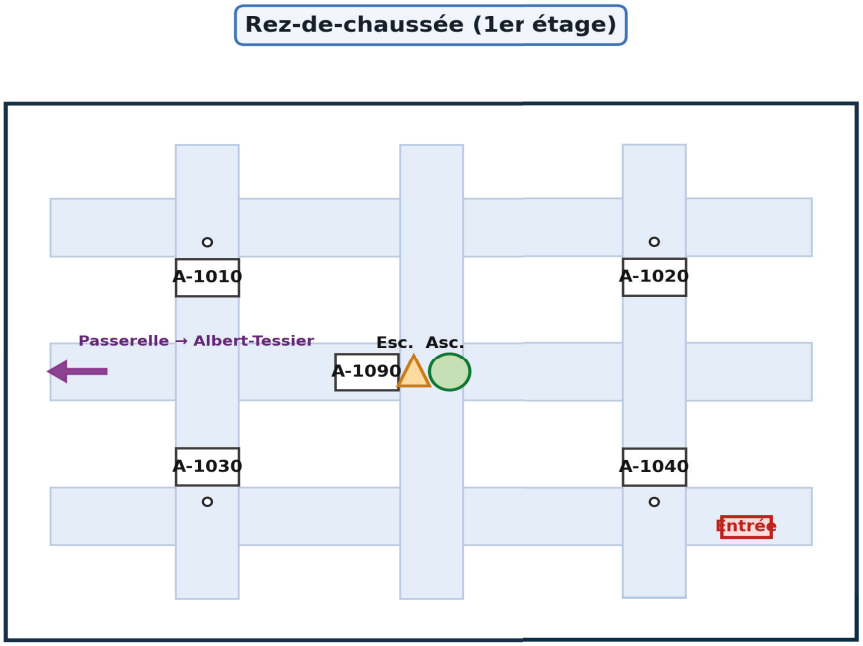

FIGURE 3.3 – Plan schématique du rez-de-chaussée (1er étage) du pavillon Ringuet. On y observe l'entrée principale (en rouge, à droite), la passerelle Albert-Tessier (flèche violette, à gauche) et cinq salles (A-1010, A-1020, A-1030, A-1040, A-1090).

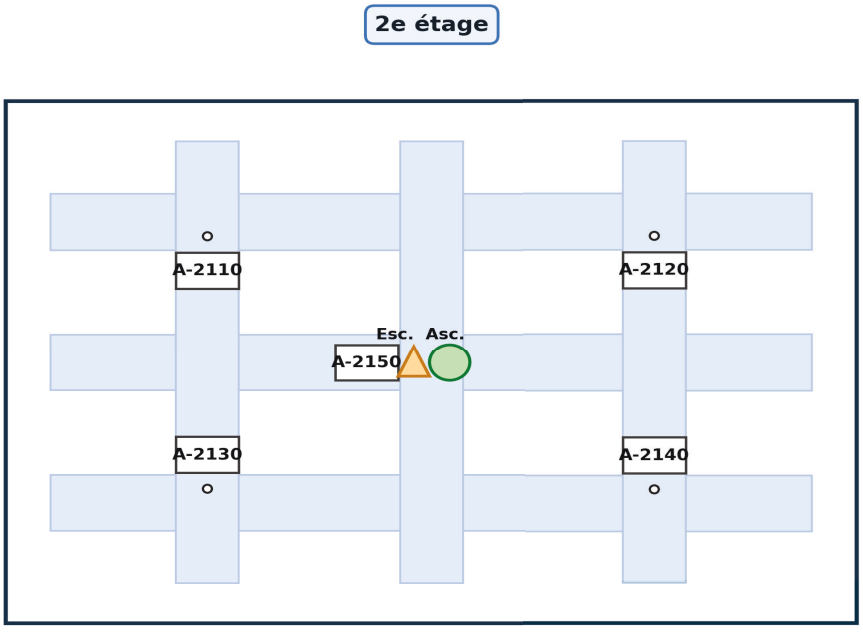

FIGURE 3.4 – Plan schématique du 2e étage du pavillon Ringuet. Cet étage comprend le laboratoire informatique A-2110 et quatre autres salles (A-2120, A-2130, A-2140, A-2150).

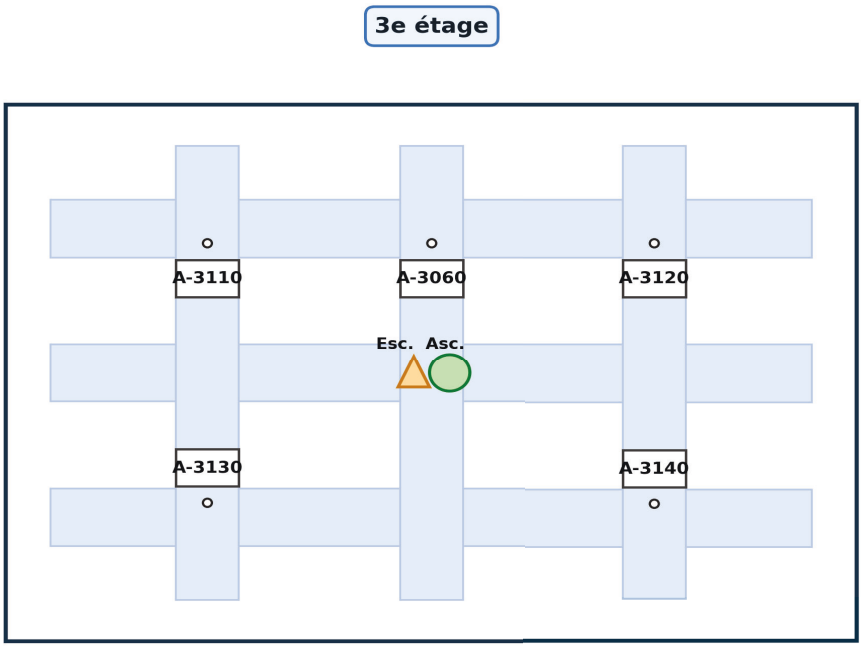

FIGURE 3.5 – Plan schématique du 3e étage du pavillon Ringuet. On y trouve la salle de réunion A-3060 ainsi que quatre salles de classe (A-3110, A-3120, A-3130, A-3140).

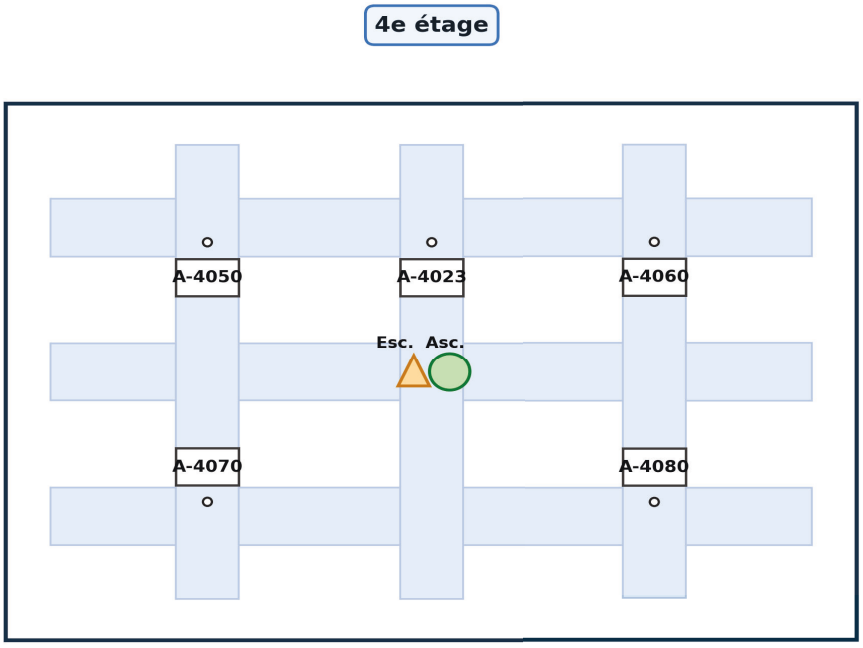

FIGURE 3.6 – Plan schématique du 4e étage du pavillon Ringuet. Cet étage accueille le laboratoire de recherche A-4023 et quatre autres salles (A-4050, A-4060, A-4070, A-4080).

Cette modélisation par étage permet de visualiser clairement la topologie du bâtiment et facilite la compréhension du fonctionnement du système de navigation. Les itiné-

raires calculés par NaviUQTR suivent ces couloirs et utilisent le noyau central pour les déplacements verticaux.

3.3.2 Représentation topologique sous forme de graphe

La modélisation spatiale développée dans ce mémoire repose sur une représentation topologique schématique d'un environnement intérieur universitaire. Cette approche privilégie la clarté pédagogique, la validation méthodologique et la généralisation plutôt que la fidélité architecturale exhaustive à un bâtiment spécifique.

Justification de l'approche schématique

Plusieurs raisons motivent le choix d'une modélisation simplifiée. D'abord, elle permet de concentrer l'évaluation sur les contributions scientifiques centrales du mémoire : l'intégration des grands modèles de langage pour la navigation intérieure, la génération d'instructions en langage naturel, et l'articulation entre planification algorithmique et interaction linguistique. Une modélisation exhaustive d'un bâtiment complexe introduirait des difficultés techniques (acquisition de plans détaillés, gestion de zones restreintes, modélisation de contraintes administratives) qui, bien que pertinentes pour un déploiement opérationnel, ne sont pas essentielles à la validation des concepts proposés.

Ensuite, l'approche schématique facilite la reproductibilité et l'extension du travail. Les chercheurs souhaitant adapter cette méthodologie à d'autres contextes peuvent directement transposer les principes développés sans dépendre de spécificités architecturales particulières. La généralisation constitue ainsi un objectif prioritaire par rapport à la spécialisation.

Enfin, cette démarche s'inscrit dans une tradition méthodologique établie en recherche sur la navigation intérieure, où les environnements simplifiés ou simulés sont couramment utilisés pour valider des concepts avant leur transposition vers des contextes réels [23], [26].

Principes de la modélisation topologique

Dans la représentation adoptée, chaque lieu significatif du modèle de bâtiment est symbolisé par un nœud du graphe. Ces nœuds correspondent à trois catégories distinctes : les intersections de couloirs, qui constituent les points de décision pour l'usager ; les salles et laboratoires, qui représentent les destinations potentielles ; et les équipements de circulation verticale, tels que les escaliers et ascenseurs. Les arêtes du graphe symbolisent les connexions physiques entre ces lieux, qu'il s'agisse de couloirs horizontaux reliant des intersections ou de liaisons verticales entre étages.

Chaque nœud du graphe est enrichi d'attributs sémantiques décrivant ses propriétés. L'attribut `type` distingue les jonctions (*junction*), les salles (*room*) et les équipements verticaux (`vertical`). Les coordonnées `x`, `y` permettent de positionner le nœud dans un plan cartésien simplifié, facilitant ainsi le calcul de distances et la visualisation. L'attribut `etage` indique le niveau du bâtiment (1 pour le rez-de-chaussée, 2 à 4 pour les étages supérieurs). Enfin, l'attribut `description` fournit une désignation textuelle utilisée pour la génération d'instructions (par exemple, « Couloir central (20,14) » ou « Laboratoire A-2110 »).

De manière similaire, chaque arête possède des attributs précisant sa nature et son coût de déplacement. L'attribut *weight* représente la distance métrique approximative, calculée selon la distance de Manhattan entre les coordonnées des nœuds connectés. L'attribut optionnel *kind* distingue les types de connexions : `corridor` pour les couloirs horizontaux, `vertical` pour les escaliers et ascenseurs, et *door* pour les passages entre couloir et salle.

Cette abstraction topologique présente plusieurs avantages par rapport à une modélisation géométrique détaillée. Elle réduit considérablement la complexité computationnelle de la recherche de chemin, tout en conservant les informations essentielles à la navigation. Elle facilite également l'intégration de contraintes sémantiques, telles que les préférences d'accessibilité ou les zones temporairement inaccessibles. Enfin, elle offre une représentation stable et compacte, facile à maintenir et à faire évoluer.

Simplifications adoptées

Plusieurs simplifications structurelles ont été introduites pour maintenir la clarté et la maniabilité du modèle.

Premièrement, la structure des couloirs suit une grille régulière de 3×3 intersections par étage, facilitant la visualisation et le calcul d'itinéraires ; cette régularité, bien que rare dans les bâtiments réels, ne limite pas la généralité de l'approche.

Deuxièmement, les codes de salles (A-1010, A-2110, A-3060, etc.) suivent une convention cohérente mais fictive, ne correspondant pas nécessairement aux numéros réels de salles existantes.

Troisièmement, le nombre de salles par étage a été limité à cinq destinations principales pour maintenir la lisibilité des démonstrations et éviter une surcharge cognitive lors de l'évaluation humaine.

Quatrièmement, les dimensions métriques sont approximatives et servent principalement à établir des distances relatives entre les nœuds ; l'échelle n'est pas rigoureusement respectée mais demeure cohérente au sein du modèle.

Enfin, certains éléments architecturaux secondaires (zones techniques, locaux de service, circulations d'urgence) ont été omis pour ne retenir que les espaces pertinents pour la navigation typique des usagers.

Transposition vers un environnement réel

Pour un déploiement opérationnel dans un bâtiment réel, une phase de modélisation détaillée serait nécessaire. Cette phase impliquerait d'abord l'acquisition des plans architecturaux officiels auprès des services d'infrastructure, puis la validation des codes de salles avec les services administratifs pour garantir leur exactitude. Il faudrait ensuite enrichir le graphe avec l'ensemble des salles, laboratoires, bureaux et espaces communs, tout en intégrant les contraintes d'accessibilité réelles (rampes, ascenseurs adaptés, portes automatiques). L'annotation des repères visuels effectivement présents dans les couloirs (œuvres d'art, panneaux d'affichage, distributeurs) viendrait compléter cette modélisation. Enfin, la prise en compte des zones à accès restreint et des horaires d'ouverture différenciés permettrait de générer des itinéraires respectant les contraintes réglementaires et temporelles.

Cette transposition ne présente pas de difficulté conceptuelle majeure, les mécanismes développés dans ce mémoire étant conçus pour s'adapter à tout graphe topologique. Le processus de modélisation détaillée constituerait essentiellement un travail d'ingénierie et de collecte de données plutôt qu'une contribution scientifique additionnelle.

Cette approche schématique offre un cadre particulièrement adapté à la planification d'itinéraires, comme l'a démontré une littérature abondante en navigation intérieure [10], [11]. Elle réduit considérablement la complexité computationnelle tout en conservant les informations essentielles à la navigation.

3.3.3 Construction du graphe multi-niveaux

Le modèle de bâtiment comporte quatre niveaux : un rez-de-chaussée (désigné comme 1^{er} étage) et trois étages supérieurs (2^{e}, 3^{e} et 4^{e}). Chaque niveau est modélisé de manière indépendante avant d'être relié verticalement au niveau du noyau central.

Pour chaque étage, la structure des couloirs suit une grille régulière de 3×3 intersections. Trois couloirs horizontaux parcourent le bâtiment d'ouest en est, positionnés aux coordonnées $y = 7$, $y = 14$ et $y = 21$. De manière symétrique, trois couloirs verticaux relient le nord au sud, aux coordonnées $x = 10$, $x = 20$ et $x = 30$. Cette géométrie simplifiée, bien qu'abstraite, facilite la compréhension des itinéraires par les usagers et permet une visualisation claire des parcours.

Les neuf intersections formées par le croisement de ces couloirs constituent les nœuds principaux du graphe pour chaque étage. Chaque intersection est représentée par un nœud de type `junction`, identifié par ses coordonnées et son numéro d'étage. Les arêtes connectant les intersections d'un même étage se voient attribuer un poids égal à la distance de Manhattan entre leurs nœuds extrémités, calculée comme $w = |x_1 - x_2| + |y_1 - y_2|$. Avec la grille régulière adoptée, les arêtes entre intersections adjacentes ont ainsi un poids de 10 unités pour les déplacements selon l'axe x (entre $x = 10$ et $x = 20$, ou entre $x = 20$ et $x = 30$), et de 7 unités pour les déplacements selon l'axe y (entre $y = 7$ et $y = 14$, ou entre $y = 14$ et $y = 21$). Ces distances relatives reflètent les proportions du modèle schématique et non des mesures métriques réelles du bâtiment.

Le noyau central du bâtiment, situé à l'intersection des couloirs médians ($x = 20$, $y = 14$), joue un rôle particulier. Il abrite les équipements de circulation verticale (escaliers et ascenseurs) permettant de passer d'un étage à l'autre. Pour modéliser ces connexions verticales, des arêtes spéciales relient les nœuds centraux de chaque paire d'étages consécutifs. Ces arêtes portent l'attribut `kind="vertical"` et se voient attribuer un poids fixe $w = 9.0$. Cette valeur a été choisie pour être comparable à la distance Manhattan moyenne entre intersections adjacentes dans le modèle (7 à 10 unités), tout en pénalisant légèrement les changements d'étage pour refléter leur coût temporel et physique plus élevé par rapport à un déplacement horizontal équivalent. Ainsi, l'algorithme de Dijkstra privilégie naturellement les déplacements horizontaux à distance égale, sans exclure les transitions verticales lorsqu'elles constituent le chemin optimal.

Le rez-de-chaussée présente deux particularités. D'une part, il comporte l'entrée principale du bâtiment, modélisée par un nœud spécifique relié au réseau de couloirs par une arête de faible poids. D'autre part, il accueille une passerelle vers un pavillon adjacent, représentée par un nœud dédié connecté au couloir ouest. Ces deux éléments constituent des points d'accès privilégiés pour les usagers entrant dans le bâtiment.

3.3.4 Intégration des points d'intérêt

Chaque étage comporte cinq salles principales, représentant des destinations potentielles pour les usagers. Ces salles sont identifiées par des codes standardisés de la forme « A-XXXX », où « A » désigne le pavillon et « XXXX » le numéro de salle. Par exemple, le laboratoire d'informatique porte le code « A-2110 » (2^{e} étage, salle 110), tandis que la salle de réunion principale est référencée « A-3060 » (3^{e} étage, salle 60).

Chaque salle est modélisée comme un nœud de type `room`, connecté à une intersection de couloir par une arête courte représentant la porte d'accès. Les attributs du nœud

incluent une description textuelle enrichie (par exemple, « Laboratoire A-2110 » plutôt que simplement « A-2110 »), facilitant ainsi la génération d'instructions naturelles.

Le positionnement géométrique des salles respecte la structure du modèle. Certaines salles sont situées au nord des couloirs horizontaux, d'autres au sud. De même, certaines se trouvent à l'est des couloirs verticaux, d'autres à l'ouest. Cette disposition influe sur la formulation des instructions : une salle au nord du couloir nécessitera une consigne du type « tournez à gauche », tandis qu'une salle au sud requerra « tournez à droite ».

Pour renforcer la clarté des instructions, chaque salle est associée à une étiquette fonctionnelle décrivant son usage. Ainsi, « A-2110 » est désigné comme « Laboratoire d'informatique », « A-3060 » comme « Salle de réunion », et « A-4023 » comme « Laboratoire de recherche ». Ces labels permettent aux usagers de formuler des requêtes en langage naturel (« Où est le laboratoire d'informatique ? ») sans nécessairement connaître les codes de salles. Cette fonctionnalité constitue précisément l'une des problématiques centrales résolues par l'affinage du modèle de langage, comme détaillé dans la section 3.5.

3.3.5 Gestion des contraintes d'accessibilité

Le système intègre plusieurs mécanismes permettant de prendre en compte les contraintes d'accessibilité et les préférences individuelles des usagers. Ces mécanismes reposent sur une pondération dynamique des arêtes du graphe en fonction du contexte de la requête.

Lorsqu'un usager indique une préférence pour l'accessibilité (par exemple, en raison d'une mobilité réduite), le système favorise les itinéraires empruntant les ascenseurs plutôt que les escaliers. Cette préférence est implémentée en réduisant le poids des arêtes de type `vertical` connectées au noyau central. Concrètement, le poids de ces arêtes est multiplié par un facteur $\alpha < 1$ (par exemple, $\alpha = 0.7$), rendant ainsi ces chemins plus attractifs pour l'algorithme de recherche.

Le système intègre également une prise en compte des heures de pointe. Durant certaines plages horaires (par exemple, entre 8h et 10h), le noyau central du rez-de-chaussée peut être fortement congestionné en raison de l'affluence des usagers. Il convient de préciser que dans le modèle actuel, un seul noyau central est présent par étage. La pénalisation ne redirige donc pas vers un noyau alternatif, mais encourage plutôt l'algorithme de Dijkstra à explorer des chemins passant par les couloirs périphériques du rez-de-chaussée avant d'atteindre le noyau, allongeant légèrement le trajet pour éviter la zone congestionnée. Dans le cas extrême où le noyau est le seul point de passage obligatoire (bâtiment à couloir unique), la pénalisation n'au-

rait aucun effet pratique. Cette limitation est reconnue et constitue une motivation supplémentaire pour l'intégration de plans architecturaux réels comportant plusieurs points de circulation verticale.

Ces mécanismes de pondération dynamique offrent une flexibilité importante et permettent d'intégrer progressivement d'autres contraintes : zones temporairement inaccessibles pour travaux, portes fermées en dehors des heures d'ouverture, ou préférences personnelles pour des parcours spécifiques. Cette approche modulaire facilite l'adaptation du système à différents contextes d'usage.

3.4 Constitution du corpus d'entraînement

La constitution d'un corpus d'entraînement adapté constitue un élément déterminant pour les performances du système. Contrairement aux approches traditionnelles d'affinage qui utilisent souvent des corpus figés, une démarche itérative d'amélioration progressive basée sur l'analyse des limitations observées à chaque version a été adoptée. Cette section présente la méthodologie de génération du corpus, puis détaille l'évolution de sa composition à travers trois versions successives.

3.4.1 Génération synthétique de conversations

En l'absence de données réelles collectées auprès d'usagers du pavillon Ringuet, une approche de génération synthétique de conversations question-réponse a été retenue. Cette méthode, couramment utilisée en apprentissage supervisé lorsque les données réelles sont insuffisantes ou indisponibles, permet de créer un corpus contrôlé et diversifié.

Le processus de génération repose sur l'énumération systématique des destinations disponibles dans le modèle de bâtiment (vingt salles réparties sur quatre étages, plus les points d'intérêt spéciaux comme la passerelle). Pour chaque destination, plusieurs formulations de questions variées sont générées, correspondant aux différentes manières dont un usager pourrait exprimer sa requête.

Chaque exemple du corpus suit le format conversationnel standard des modèles de type *instruction-tuning*, avec deux rôles distincts : `user` pour la question et `assistant` pour la réponse attendue. La question peut prendre plusieurs formes syntaxiques selon le contexte et les informations disponibles à l'usager. Les questions directes avec code de salle incluent des formulations comme « Où est A-2110 ? » ou « Où se trouve la salle A-2110 ? ». Les questions de navigation avec code utilisent plutôt « Comment aller à A-3060 ? » ou « Itinéraire vers A-4023 ». Les questions avec *alias* sémantiques permettent de chercher par fonction, par exemple « Où est le laboratoire informatique ? » ou «

Je cherche la salle de réunion ». Enfin, les questions mixtes combinent navigation et *alias*, comme dans « Comment aller au laboratoire de recherche ? ».

La réponse attendue est systématiquement le code de la salle cible (par exemple, « A-2110 »), sans information supplémentaire. Cette approche minimaliste vise à ce que le modèle se concentre exclusivement sur la tâche d'identification de destination, laissant les étapes ultérieures du pipeline gérer le calcul d'itinéraire et la génération d'instructions détaillées.

Cette conception peut sembler redondante pour les requêtes contenant explicitement un code de salle (par exemple, « Où est A-2110 ? »), puisque la réponse attendue (A-2110) est déjà présente dans la question. Il convient de préciser que ces exemples ne constituent pas la raison d'être principale du module LLM : ils sont inclus dans le corpus pour assurer la robustesse du modèle face aux formulations directes, mais leur traitement est en pratique délégué au module de *pattern matching* (Module 2A) qui court-circuite le LLM dès qu'un code valide est détecté. La valeur ajoutée du LLM réside exclusivement dans le traitement des requêtes sans code explicite, où l'utilisateur décrit sa destination par sa fonction (« laboratoire informatique », « salle de réunion ») sans connaître le code correspondant. Un système de règles simple serait effectivement suffisant pour les requêtes directes, mais serait incapable de mapper des descriptions fonctionnelles vers des codes techniques sans une compréhension sémantique que seul un modèle de langage peut fournir de manière générale. Il est important de souligner que pour les requêtes contenant uniquement des alias connus et prédéfinis, un système de règles simple (dictionnaire de correspondances alias→code) serait effectivement plus efficace et plus prévisible. La valeur du LLM réside dans sa capacité à généraliser à des formulations non vues durant l'entraînement : des variantes syntaxiques, des abréviations, des descriptions partielles ou des formulations informelles que ni un dictionnaire ni des règles régulières ne pourraient anticiper exhaustivement. Cette capacité de généralisation sémantique constitue la justification principale du choix d'un LLM pour cette tâche.

Le corpus est ensuite divisé en deux sous-ensembles selon une répartition 70/30 : un ensemble d'entraînement (*train*) utilisé pour l'affinage proprement dit, et un ensemble de validation (*validation*) permettant de surveiller la convergence et d'éviter le surapprentissage. Cette division est effectuée de manière stratifiée pour garantir que chaque destination soit représentée dans les deux ensembles avec des proportions similaires.

3.4.2 Stratégie itérative d'amélioration du corpus

L'un des apports méthodologiques majeurs de ce travail réside dans l'adoption d'une approche itérative pour optimiser la composition du corpus d'entraînement. Plutôt

que de considérer la constitution du corpus comme une étape préliminaire figée, le choix a été fait de l'adapter progressivement en fonction des résultats d'évaluation obtenus. Cette démarche empirique s'inspire des meilleures pratiques en apprentissage automatique, où l'analyse des erreurs guide l'amélioration des données d'entraînement.

Trois versions successives du corpus ont été développées, chacune répondant à des limitations spécifiques identifiées lors de l'évaluation de la version précédente. La Figure 3.7 illustre l'évolution de la précision du système à travers ces trois itérations.

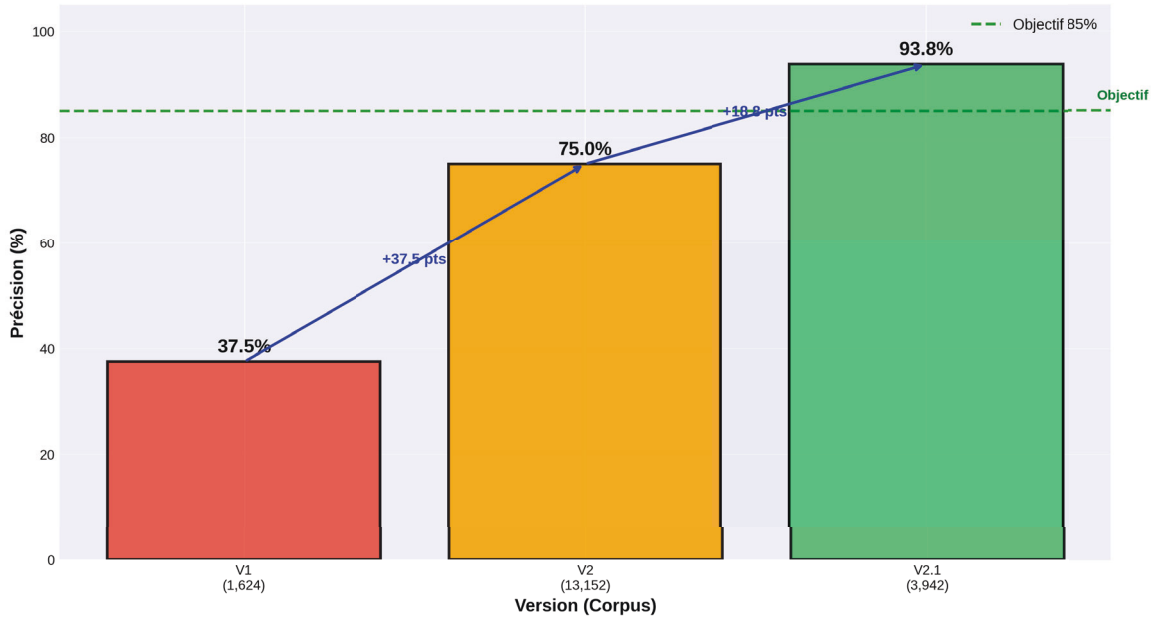

FIGURE 3.7 – Évolution de la précision du système NaviUQTR à travers les versions V1, V2 et V2.1. L'amélioration progressive résulte d'ajustements ciblés de la composition du corpus d'entraînement.

Note : Le seuil de 85% utilisé comme objectif de référence dans cette figure correspond au niveau de performance minimal rapporté dans les systèmes comparables de la littérature, notamment MapGPT [2] (85–90%) et Zhang et al. [3] (88–92%). Ce seuil a été retenu comme repère comparatif, sans constituer une contrainte formelle de conception.

Version 1 (V1) : Corpus exploratoire initial

La version V1 a été développée comme point de départ exploratoire de la démarche itérative, sans contrainte sur la composition du corpus. On pourrait objecter que développer une version avec un corpus biaisé est inutile si ce biais est prévisible. Cependant, cette étape exploratoire remplit trois fonctions méthodologiques importantes. Premièrement, elle établit une *baseline* de référence permettant de mesurer le gain apporté par chaque amélioration successive. Deuxièmement, elle révèle em-

piriquement quels types de biais émergent naturellement lors d'une génération non contrôlée, information qui n'était pas évidente a priori. Troisièmement, elle valide le pipeline technique complet (génération du corpus, affinage LoRA, évaluation) avant d'investir dans une stratégie de composition plus élaborée. Cette démarche empirique, bien que plus longue qu'une optimisation directe, s'inscrit dans les meilleures pratiques de l'apprentissage automatique itératif guidé par l'analyse des erreurs.

La première version du corpus, désignée par V1, comportait 1 624 exemples d'entraînement générés selon la méthodologie décrite précédemment. Lors de la génération initiale, aucune attention particulière n'avait été accordée à l'équilibre entre les différents types de questions. L'analyse a révélé une distribution fortement déséquilibrée dans la composition du corpus. La formulation « Comment aller à... ? » dominait avec environ 60% des exemples, tandis que les questions d'identification directe « Où est... ? » ou « Où se trouve... ? » ne représentaient qu'environ 9% du total. Plus problématique encore, les *alias* sémantiques (« laboratoire informatique », « salle de réunion ») constituaient moins de 5% du corpus. Le reste se répartissait entre diverses formulations mineures sans impact significatif sur les performances.

Cette composition déséquilibrée a eu des conséquences directes sur les performances du modèle affiné. L'évaluation de V1 a révélé une précision globale de seulement 37,5% sur un ensemble de tests couvrant différents types de questions. L'analyse détaillée des résultats, présentée dans la Figure 3.8, montre que le modèle échouait complètement (0% de réussite) sur les questions d'identification directe du type « Où est A-2110 ? », alors qu'il réussissait parfaitement (100%) les questions de navigation « Comment aller à A-2110 ? ».

Ce constat s'explique par le biais induit par la composition du corpus : le modèle avait appris à associer systématiquement toute question à une requête de navigation, rendant les formulations minoritaires inefficaces. Concernant les *alias* sémantiques, le résultat de 100% en V1 doit être interprété avec prudence : il repose sur seulement 3 exemples de test issus d'un corpus d'entraînement lui-même pauvre en *alias*, ce qui soulève des questions légitimes sur la capacité de généralisation réelle du modèle. Cette fragilité deviendra évidente lors de l'évaluation de V2.

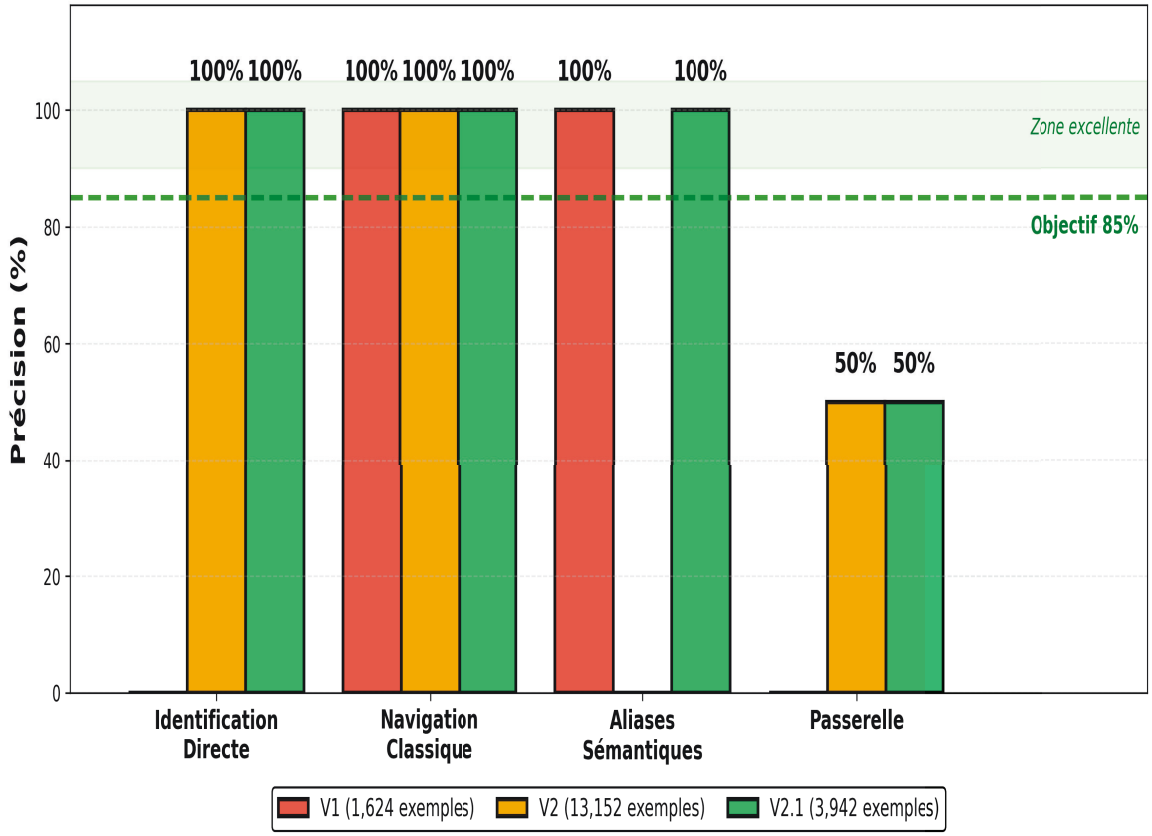

FIGURE 3.8 – Comparaison des performances par type de tâche à travers les versions V1, V2 et V2.1. Les limitations de chaque version orientent les améliorations de la suivante.

Version 2 (V2) : Augmentation massive du corpus

Face aux limitations de V1, une deuxième version du corpus (V2) privilégiant une augmentation substantielle du nombre d'exemples a été développée. L'hypothèse formulée était qu'un corpus plus large permettrait au modèle de mieux généraliser et de corriger les biais observés en V1.

La génération de V2 a suivi un processus d'énumération systématique et combinatoire visant à maximiser la couverture des formulations possibles. Pour chacune des 20 destinations, toutes les combinaisons entre 6 formulations de questions (« Où est », « Où se trouve », « C'est où », « Comment aller », « Itinéraire vers », « Je cherche ») et 3 variantes syntaxiques (avec/sans article, avec/sans point d'interrogation, formes condensées) ont été générées. Ce corpus de base a ensuite été enrichi en ajoutant des variations orthographiques et de casse pour augmenter la robustesse du modèle face aux erreurs de frappe potentielles. Enfin, des formulations avec codes de salles incomplets (« A 2110 » au lieu de « A-2110 ») ont été introduites pour gérer les cas où l'usager omet le tiret séparateur.

Cette approche a produit un corpus de 13 152 exemples d'entraînement (après division train/validation : 9 206 pour l'entraînement, 3 946 pour la validation), soit une augmentation d'un facteur 8 par rapport à V1.

Une précision s'impose concernant le risque de chevauchement entre les ensembles d'entraînement et de validation. La division stratifiée 70/30 porte sur les *destinations* et non sur les formulations individuelles : pour chaque destination, 70% des exemples sont alloués à l'entraînement et 30% à la validation. Cela garantit que chaque destination est représentée dans les deux sous-ensembles, mais implique que des formulations similaires peuvent apparaître dans les deux sous-ensembles pour une même destination, en particulier dans le cas d'une énumération combinatoire exhaustive comme celle adoptée en V2. Ce risque de contamination partielle est une limitation reconnue de l'approche par génération synthétique exhaustive. En V2.1, la réduction du corpus et la diversification contrôlée des formulations limitent ce phénomène. Une validation sur des requêtes réelles d'utilisateurs constituerait une évaluation plus rigoureuse, identifiée comme perspective à court terme.

L'évaluation de V2 a montré une amélioration significative de la précision globale, passant de 37,5% à 75,0%. Le problème d'identification directe avait été complètement résolu : le modèle atteignait désormais 100% de réussite sur les questions « Où est A-2110 ? », contre 0% en V1. Les questions de navigation maintenaient également une performance parfaite à 100%.

Cependant, un nouveau problème critique est apparu : la précision sur les *alias* sémantiques a chuté à 0%, alors qu'elle était à 100% en V1. L'analyse du corpus V2 a révélé la cause de cette régression : malgré l'augmentation massive du nombre total d'exemples, la proportion d'*alias* sémantiques était restée très faible (moins de 5% du corpus). Le phénomène de dilution avait joué en défaveur de cette catégorie : les exemples d'*alias*, bien qu'en nombre absolu légèrement supérieur à V1, étaient noyés dans la masse des exemples utilisant des codes de salles explicites.

Ce constat illustre un principe important en apprentissage automatique découvert empiriquement : l'augmentation quantitative du corpus ne garantit pas automatiquement une amélioration qualitative si la composition relative des différentes catégories n'est pas soigneusement contrôlée. La Figure 3.9 met en évidence cette problématique spécifique.

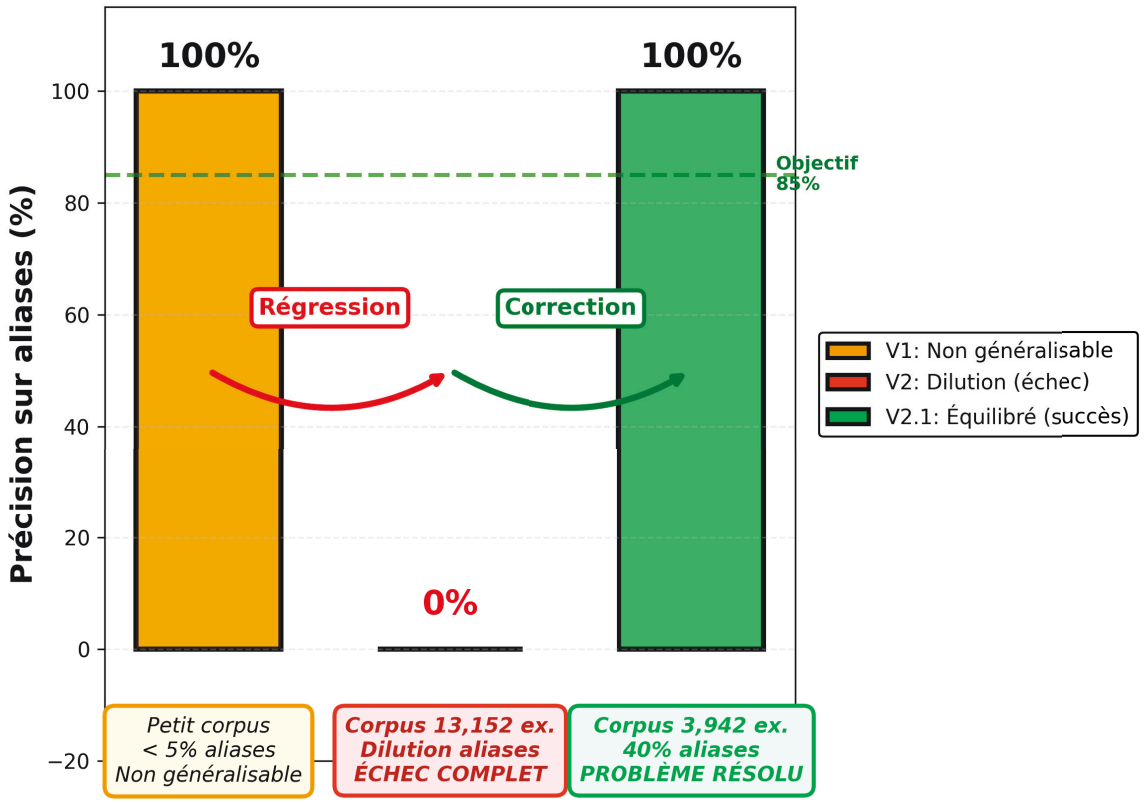

FIGURE 3.9 – Évolution de la performance sur les *alias* sémantiques. V2 subit une dilution malgré l'augmentation du corpus, problème résolu en V2.1 par un rééquilibrage ciblé.

Version 2.1 (V2.1) : Équilibrage qualitatif du corpus

La troisième et dernière version du corpus (V2.1) adopte une approche radicalement différente, privilégiant l'équilibrage qualitatif plutôt que la maximisation quantitative. Plutôt que d'augmenter encore le nombre total d'exemples, le choix a été fait de réduire la taille du corpus tout en garantissant une représentation équilibrée de chaque catégorie de questions. La stratégie de composition de V2.1 repose sur quatre principes fondamentaux. D'abord, une représentation équilibrée des alias où 40% des exemples utilisent des formulations avec alias sémantiques (« laboratoire informatique », « salle de réunion », etc.). Ensuite, le maintien de codes explicites avec 60% des exemples conservant l'utilisation directe de codes de salles. Par ailleurs, la diversité syntaxique est préservée grâce au maintien de la variété des formulations (« Où est », « Comment aller », etc.). Enfin, la taille a été optimisée pour produire un corpus final de 3 942 exemples (2 768 entraînement, 1 174 validation).

Cette composition équilibrée représente un compromis informé par les observations des versions précédentes. La proportion de 40% d'alias garantit une exposition suffisante du modèle à ce type de formulation, tout en conservant une majorité de codes explicites pour maintenir la performance sur les requêtes directes.

L'évaluation de V2.1 a confirmé le bien-fondé de cette approche. La précision globale a atteint 93,8%, soit une amélioration de 18,8 points par rapport à V2 et de 56,2 points par rapport à V1.

Plus significativement encore, toutes les catégories de questions atteignent désormais des performances excellentes. L'identification directe atteint 100% de réussite, performance maintenue depuis V2. La navigation classique conserve également 100% de réussite, stabilité observée depuis V1. Les alias sémantiques, problème majeur de V2, sont désormais résolus avec 100% de réussite. Seule la catégorie passerelle présente une faiblesse résiduelle à 50%, dont l'impact demeure cependant limité.

Le Tableau 3.1 synthétise les caractéristiques des trois versions du corpus et leurs performances respectives.

TABLE 3.1 – Comparaison des trois versions du corpus d'entraînement

Caractéristique	**V1**	**V2**	**V2.1**
Taille (train)	1,624	9,206	2,768
Composition	Déséquilibrée	Codes explicites	Équilibrée
alias (%)	$< 5\%$	$< 5\%$	40%
Précision globale	37,5%	75,0%	93,8%
Identification	0%	100%	100%
Navigation	100%	100%	100%
alias	100%*	0%	100%

* Petit corpus, faible généralisation

Cette progression itérative illustre un principe méthodologique important : l'optimisation d'un corpus d'entraînement nécessite un processus empirique d'ajustements successifs guidé par l'analyse des performances. La simple augmentation quantitative (V1 $\rightarrow$ V2) ne suffit pas ; un équilibrage qualitatif réfléchi (V2 $\rightarrow$ V2.1) s'avère déterminant pour atteindre des performances élevées sur l'ensemble des catégories de tâches.

3.5 Adaptation du modèle de langage

L'adaptation du modèle de langage au domaine spécifique de la navigation intérieure constitue le cœur technique de ce mémoire. Cette section présente les choix méthodologiques concernant le modèle de base, la technique d'affinage, et la stratégie d'entraînement.

3.5.1 Choix du modèle de base

Le choix du modèle de langage constitue une décision stratégique conditionnant les performances et les contraintes d'implémentation du système. Plusieurs critères ont guidé la sélection : la capacité de compréhension du langage naturel, la facilité d'adaptation à un domaine spécialisé, la disponibilité en source ouverte, et la qualité de la génération textuelle.

Après évaluation de plusieurs alternatives (GPT-3, PaLM, LLaMA-2, Mistral), le modèle **Mistral-7B-Instruct-v0.2** a été retenu pour plusieurs raisons. D'abord, sa taille intermédiaire (7 milliards de paramètres) offre un bon compromis entre performance et efficacité computationnelle. Ensuite, sa licence source ouverte (Apache 2.0) facilite l'expérimentation et l'adaptation sans contraintes commerciales. De plus, la version « Instruct » a été pré-entraînée pour suivre des instructions, ce qui correspond bien à la tâche d'identification de destination visée.

Mistral-7B présente des capacités de raisonnement spatial documentées dans plusieurs travaux récents [37], ce qui en fait un candidat particulièrement adapté à la navigation intérieure. De plus, sa structure d'attention permet de traiter efficacement des contextes longs, une propriété utile pour la génération d'itinéraires complexes comportant de nombreuses étapes.

Enfin, la disponibilité du modèle sur la plateforme Hugging Face facilite son déploiement et son intégration avec les outils standard d'affinage utilisés.

3.5.2 Configuration LoRA pour l'efficacité de l'affinage

L'affinage complet d'un modèle de 7 milliards de paramètres nécessite des ressources computationnelles considérables et présente un risque élevé de *catastrophic forgetting* (oubli catastrophique des connaissances générales). Pour pallier ces limitations, la technique **LoRA** (*Low-Rank Adaptation*) [42] a été adoptée, technique qui permet un affinage efficace en ne modifiant qu'une fraction des paramètres du modèle.

LoRA introduit des matrices de rang faible qui sont entraînées pendant l'affinage, tandis que les poids originaux du modèle restent figés. Cette approche présente plusieurs avantages : réduction drastique du nombre de paramètres à entraîner (typiquement 0,1% à 3% du total), diminution des besoins en mémoire GPU, et préservation des capacités générales du modèle pré-entraîné.

La configuration LoRA retenue repose sur plusieurs hyperparamètres soigneusement choisis. Le **rang r** $= 64$ contrôle la capacité d'adaptation des matrices LoRA ; un rang plus élevé offre plus de flexibilité mais augmente le nombre de paramètres entraînables. Le facteur **alpha** $\boldsymbol{\alpha} = 128$ constitue un facteur de mise à l'échelle qui

contrôle l'influence des adaptateurs LoRA, la valeur $\alpha = 2r$ étant une heuristique courante. Concernant les **modules cibles**, LoRA est appliqué aux projections d'attention (q_proj, k_proj, v_proj, o_proj) et aux couches feed-forward (gate_proj, up_proj, down_proj), couvrant ainsi l'ensemble des transformations linéaires du modèle. Enfin, un **dropout** de 0,1 assure la régularisation pour éviter le surapprentissage sur le corpus d'entraînement.

Avec cette configuration, le nombre de paramètres entraînables s'élève à environ 157 millions sur un total de 7 milliards, soit 2,26% du modèle complet. Cette efficacité paramétrique permet un entraînement rapide (1 à 3 heures selon la version du corpus) sur un GPU A100, tout en maintenant d'excellentes performances.

3.5.3 Stratégie d'affinage itératif

L'entraînement du modèle suit une procédure standardisée basée sur l'optimisation par descente de gradient stochastique. L'optimiseur AdamW est utilisé avec un taux d'apprentissage de 1×10^{-4}, une stratégie de warm-up linéaire sur 10% des étapes, et un scheduler cosinus pour la décroissance progressive du taux d'apprentissage.

Les autres hyperparamètres d'entraînement ont été configurés de manière standard. Le **nombre d'époques** a été fixé à 5 pour toutes les versions, garantissant une convergence suffisante sans surapprentissage excessif. Le **batch size effectif** de 16 (1 par device $\times$ 16 gradient accumulation steps) permet un entraînement stable malgré les contraintes mémoire. La **précision** FP16 (float 16 bits) accélère l'entraînement tout en préservant la qualité des gradients. La **longueur maximale** de 512 tokens s'avère largement suffisante pour les questions courtes typiques de la navigation. La **stratégie d'évaluation** consiste en une évaluation tous les 500 *steps* sur l'ensemble de validation, permettant de surveiller la convergence. Enfin, seul le meilleur modèle selon la perte de validation est **sauvegardé**, implémentant ainsi un *early stopping* implicite.

Comme détaillé dans la section 3.4, trois versions successives du modèle correspondant aux trois versions du corpus ont été entraînées. Cette approche itérative a permis d'identifier et de corriger progressivement les limitations du système. Le Tableau 3.2 présente les métriques d'entraînement obtenues pour chaque version. Un indicateur particulièrement révélateur est le ratio validation/entraînement : V2.1 présente le meilleur ratio (1,10), indiquant un équilibre optimal entre spécialisation sur le domaine cible et capacité de généralisation.

TABLE 3.2 – Métriques d'entraînement des trois versions du modèle

Métrique	**V1**	**V2**	**V2.1**
Perte d'entraînement	0,092	0,186	0,258
Perte de validation	0,135	0,315	0,284
Ratio val/train	1,47	1,69	**1,10**
Durée (A100-80GB)	59 min	3h53	1h06
Steps totaux	508	2,432	865

Note : *Les pertes d'entraînement et de validation ne sont pas directement comparables entre versions, car chaque version est entraînée sur un corpus différent en taille et en composition. La perte absolue d'un modèle dépend de la difficulté intrinsèque de son corpus d'entraînement : un corpus plus diversifié (V2.1) produit naturellement une perte d'entraînement plus élevée qu'un corpus homogène (V1), sans que cela indique une moins bonne performance. C'est pourquoi le ratio val/train est utilisé comme indicateur comparatif relatif plutôt que les valeurs absolues de perte. La performance finale sur l'ensemble de test (précision globale) constitue la mesure de référence absolue.*

L'évolution de ces métriques reflète les ajustements successifs du corpus. V1, avec son corpus déséquilibré, converge très rapidement vers une perte faible mais souffre d'un surapprentissage modéré (ratio 1,47). V2, malgré son corpus massif, présente un ratio encore plus élevé (1,69), suggérant une spécialisation excessive sur les codes explicites au détriment des alias. V2.1, avec son corpus équilibré, atteint le meilleur compromis : la perte d'entraînement est certes plus élevée (le modèle n'a pas « mémorisé » les exemples), mais la perte de validation est comparable, et surtout le ratio 1,10 témoigne d'une excellente généralisation.

La Figure 3.10 visualise ces métriques et met en évidence l'amélioration progressive du ratio validation/entraînement à travers les trois versions. Le panneau A présente les pertes absolues d'entraînement et de validation. Le panneau B présente le ratio validation/entraînement utilisé comme indicateur comparatif relatif entre versions. Le seuil et la zone verte sont des repères empiriques propres à cette étude, non des valeurs universelles issues de la littérature. Ces deux panneaux doivent être interprétés conjointement.

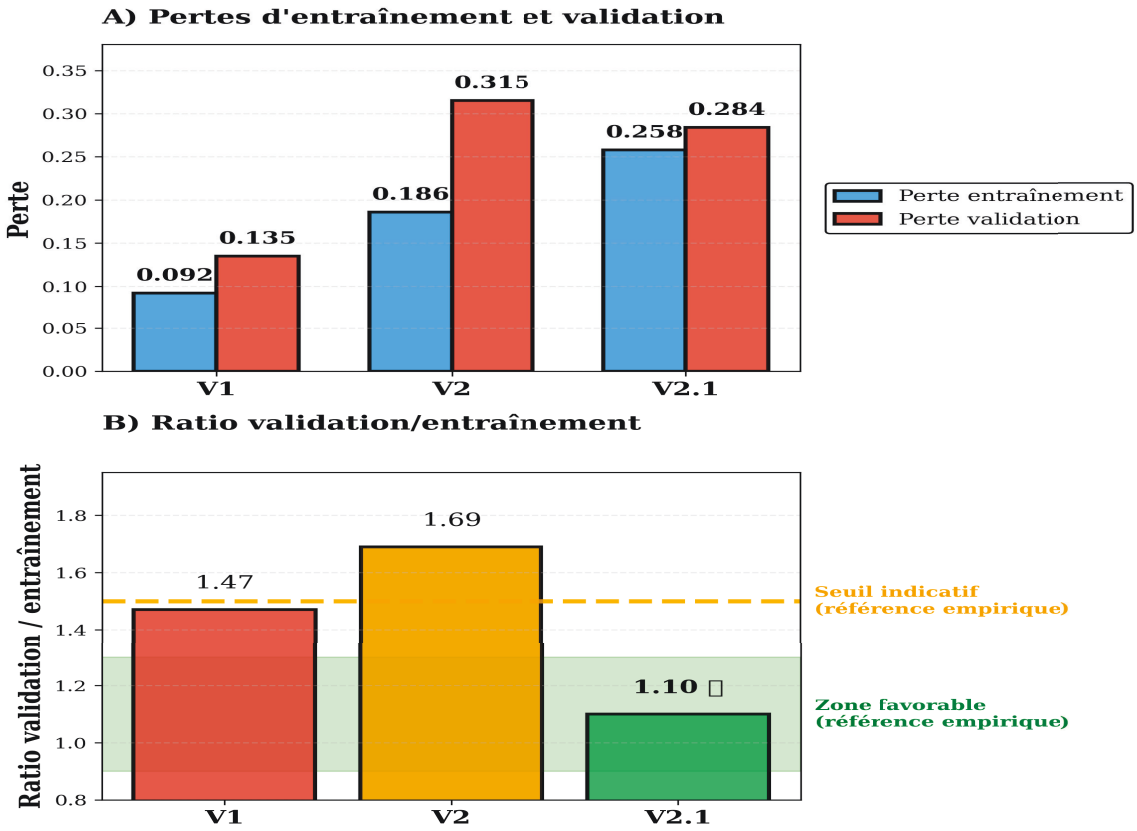

FIGURE 3.10 – Comparaison des métriques d'entraînement (perte et ratio validation/entraînement) des trois versions. V2.1 présente le meilleur équilibre.

Cette progression méthodique illustre l'importance d'une approche itérative en apprentissage automatique : plutôt que de chercher la configuration optimale dès la première tentative, une démarche empirique d'ajustements successifs guidée par l'analyse des résultats a été privilégiée. Cette stratégie s'est révélée particulièrement efficace pour identifier les problèmes subtils (comme la dilution des *alias* en V2) qui n'auraient pas été évidents a priori.

Il convient de préciser la nature et les limites de l'indicateur présenté dans le panneau B. Le ratio validation/entraînement est utilisé ici comme outil comparatif empirique entre les trois versions, et non comme une métrique standard universellement établie dans la littérature. Un ratio proche de 1,0 ne constitue pas en soi une garantie de qualité : si les deux pertes absolues sont élevées, le modèle n'a simplement pas convergé, indépendamment de leur rapport. C'est pourquoi les panneaux A et B doivent être lus conjointement. Dans notre cas, le panneau A confirme que les pertes absolues des trois versions demeurent dans des plages raisonnables pour une tâche d'instruction-tuning sur corpus spécialisé [42], ce qui donne du sens à la comparaison des ratios dans le panneau B. La précision finale sur l'ensemble de test demeure la métrique de référence absolue pour évaluer les performances du système.

3.6 Pipeline de navigation

Le pipeline de navigation intègre l'ensemble des modules présentés précédemment dans un flux de traitement cohérent. Cette section détaille le fonctionnement de chaque étape, de la capture de la requête utilisateur jusqu'à la génération des instructions finales.

3.6.1 Interprétation des requêtes en langage naturel

La première étape du pipeline consiste à analyser la requête formulée par l'usager pour préparer son traitement par le module LLM. Bien que le modèle affiné soit capable de traiter directement des requêtes en langage naturel, une phase préliminaire de correspondance de motifs (*pattern matching*) a été intégrée pour améliorer la robustesse du système.

Le module d'interprétation applique plusieurs heuristiques simples basées sur des expressions régulières. La **détection de codes de salles** recherche des motifs correspondant au format « A-XXXX » ou « A XXXX » (avec ou sans tiret), permettant d'identifier immédiatement les requêtes explicites. La **normalisation orthographique** effectue une conversion en majuscules et supprime les espaces superflus pour standardiser les entrées. Enfin, la **détection de termes spéciaux** identifie des mots-clés comme « passerelle », « entrée principale », ou « noyau central » qui correspondent à des destinations particulières du système.

Cette phase de pré-traitement, bien que simple, permet de filtrer rapidement certaines requêtes sans solliciter le modèle LLM. Par exemple, si la requête contient explicitement un code valide comme « A-2110 », le système peut court-circuiter l'appel au LLM et passer directement à l'étape suivante. Cependant, pour garantir la robustesse, toute requête ambiguë ou ne correspondant à aucun motif reconnu est systématiquement soumise au module LLM.

3.6.2 Identification de destination par LLM affiné

Le cœur du système réside dans l'utilisation du modèle Mistral-7B affiné pour identifier la destination souhaitée. Contrairement aux approches traditionnelles basées uniquement sur des règles ou des correspondances exactes, cette approche permet de gérer efficacement la variabilité linguistique et les formulations indirectes.

Le processus d'identification suit un enchaînement logique de quatre étapes complémentaires. La première étape, le **formatage de la requête**, consiste à encapsuler la question de l'usager dans le format conversationnel attendu par Mistral,

par exemple : `<s>[INST] Où est le laboratoire informatique ? [/INST]`. La deuxième étape, la **génération de la réponse**, fait appel au modèle affiné qui produit un code de salle (ex : « A-2110 ») ; les paramètres de génération sont configurés pour favoriser le déterminisme, avec une température fixée à 0,05 pour une génération quasi-déterministe minimisant la variabilité, un paramètre *top-p* réglé à 0,95 pour conserver une légère variabilité tout en concentrant la probabilité sur les *tokens* les plus pertinents, et *max_new_tokens* limité à 80, valeur largement suffisante pour générer un code court. La troisième étape, l'**extraction du code**, analyse la réponse générée par expression régulière pour extraire le code de salle au format « A-XXXX ». Enfin, la quatrième étape, la **validation**, vérifie que le code extrait correspond bien à une destination valide dans le graphe topologique. Lorsque la validation échoue, soit parce que le LLM n'a pas généré de code au format attendu, soit parce que le code généré ne correspond à aucune destination connue dans le graphe, le système retourne un message d'erreur explicite invitant l'usager à reformuler sa requête : *« Destination non reconnue. Veuillez préciser votre destination en utilisant un code de salle (exemple : A-2110) ou une description fonctionnelle reconnue (exemple : laboratoire informatique). »* Ce mécanisme de repli garantit que le système ne génère jamais d'itinéraire vers une destination invalide, préservant ainsi la fiabilité du pipeline complet. Une amélioration future consisterait à implémenter un mécanisme de clarification interactif permettant au système de proposer des destinations candidates proches de la requête non reconnue.

Apprentissage du format de sortie via le corpus

Le modèle Mistral-7B affiné n'utilise pas de invite système (*prompt* explicite pour contraindre le format de sortie. Au lieu de cela, le format structuré (codes de salles au format « A-XXXX ») est appris implicitement à travers les 3 942 exemples du corpus d'entraînement V2.1. Chaque exemple du corpus associe une requête en langage naturel à une réponse systématiquement formatée comme un code de salle simple, sans texte additionnel. Cette cohérence absolue dans le corpus permet au modèle de généraliser ce pattern lors de l'inférence, produisant des sorties structurées de manière prévisible.

Cette approche présente plusieurs avantages méthodologiques. D'abord, elle évite la complexité d'ingénierie associée à la conception de prompts système détaillés. Ensuite, elle permet au modèle d'apprendre naturellement les patterns attendus à partir des données d'entraînement. Enfin, elle garantit une cohérence entre l'entraînement et l'inférence, puisque le modèle génère exactement le type de sortie qu'il a observé pendant l'affinage.

L'efficacité de cette stratégie est confirmée par les résultats expérimentaux : le modèle V2.1 atteint 93,8% de précision sur l'ensemble de test, démontrant sa capacité à générer de manière fiable des codes de salles correctement formatés sans guidage explicite par invite système (*prompt*.

Cette flexibilité améliore significativement l'expérience utilisateur par rapport à un système basé uniquement sur des mots-clés prédéfinis.

3.6.3 Calcul d'itinéraires optimaux

Une fois la destination identifiée, le système calcule l'itinéraire optimal reliant l'origine (généralement l'entrée principale ou la position actuelle de l'usager) à cette destination. Cette tâche repose sur l'algorithme de recherche de plus court chemin de Dijkstra [19], appliqué au graphe topologique du bâtiment.

L'algorithme de Dijkstra garantit l'optimalité du chemin calculé en fonction de la métrique de poids définie sur les arêtes. Dans l'implémentation proposée, cette métrique correspond à une distance approximative reflétant le temps de parcours estimé. Les arêtes horizontales et verticales dans les couloirs ont des poids proportionnels à la distance de Manhattan entre leurs extrémités, tandis que les arêtes verticales (escaliers/ascenseurs) se voient attribuer un poids supérieur (9,0) pour refléter le coût temporel d'un changement d'étage.

L'algorithme procède par exploration progressive du graphe, en maintenant pour chaque nœud une estimation de la distance minimale depuis l'origine. À chaque itération, le nœud non visité ayant la plus faible distance estimée est sélectionné, et ses voisins sont mis à jour. Ce processus se poursuit jusqu'à ce que le nœud de destination soit atteint.

Le résultat de l'algorithme est une séquence ordonnée de nœuds formant le chemin optimal :

```
[ENTREE, JONCTION(10,14,1), NOYAU(20,14,1),
NOYAU(20,14,2), JONCTION(20,21,2), A-2110]
```

La Figure 3.11 présente la visualisation d'un itinéraire optimal généré par NaviU-QTR en réponse à la requête « Comment rejoindre le pavillon Albert-Tessier ? ». L'algorithme de Dijkstra identifie le chemin de coût minimal depuis le nœud d'entrée principale ($N_{34,6.5,1}$) jusqu'au nœud de la passerelle ($N_{4,14,1}$), en traversant successivement les intersections du réseau de couloirs du rez-de-chaussée. Le tracé suit la métrique de distance Manhattan sur le graphe topologique, garantissant l'optimalité du parcours en termes de distance cumulée. Les segments colorés et les flèches di-

rectionnelles indiquent la séquence de déplacements à effectuer le long des arêtes du graphe.

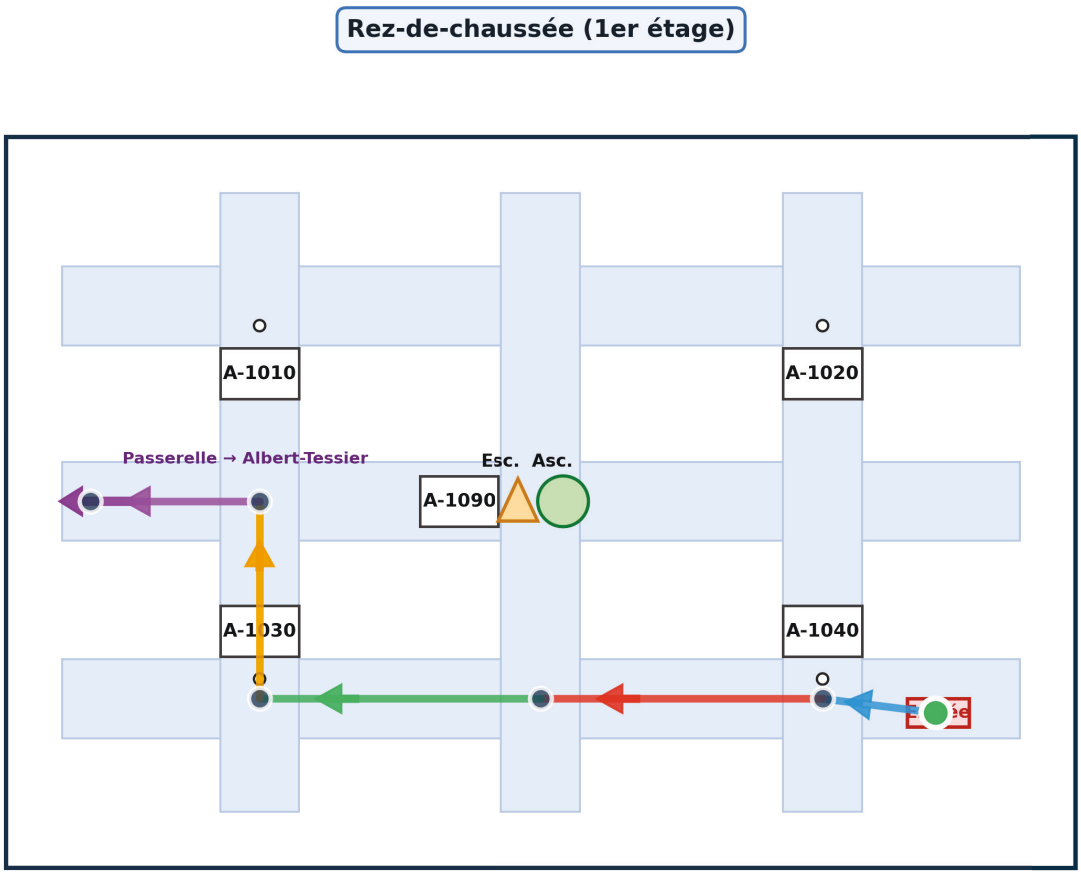

FIGURE 3.11 – Itinéraire optimal calculé par Dijkstra pour la requête « Comment rejoindre le pavillon Albert-Tessier ? ».

Cette séquence est ensuite transmise au module de génération d'instructions pour être transformée en consignes verbales.

Pour tenir compte des préférences individuelles et des contraintes contextuelles, le système peut modifier dynamiquement les poids des arêtes avant d'exécuter l'algorithme. Par exemple, si l'usager indique une préférence pour l'accessibilité, le poids des arêtes correspondant aux ascenseurs est réduit (multiplication par 0,7), favorisant ainsi ces équipements par rapport aux escaliers.

3.6.4 Génération d'instructions contextualisées

La transformation d'une séquence de nœuds en instructions verbales constitue une étape cruciale pour l'utilisabilité du système. Les consignes doivent être claires, concises, et adaptées au profil de l'usager.

Le générateur d'instructions développé suit une logique structurée en plusieurs phases. D'abord, il identifie les transitions significatives dans la séquence de nœuds : changements de direction à une intersection, changements d'étage, entrée dans une salle. Ensuite, pour chaque transition, il génère une consigne verbale appropriée en fonction du contexte.

Les consignes de base suivent des modèles prédéfinis adaptés au type de transition. Pour un déplacement simple le long d'un couloir, l'instruction prend la forme « Suivez le couloir vers l'est ». Lors d'un changement de direction, la consigne devient « À l'intersection, tournez à gauche ». Pour un changement d'étage, le système génère « Prenez l'ascenseur et montez au 2e étage ». Enfin, pour l'entrée dans la destination finale, l'instruction indique « La salle A-2110 se trouve sur votre gauche ».

Ces modèles de base sont enrichis d'informations contextuelles pour améliorer leur clarté. Par exemple, plutôt que simplement « tournez à gauche », le système peut générer « À l'intersection, tournez à gauche en direction du laboratoire A-2110 », fournissant ainsi à l'usager une confirmation de la direction correcte.

Le générateur prend également en compte l'étage de départ et d'arrivée pour formuler des consignes adaptées. Au rez-de-chaussée, le système peut faire référence à l'entrée principale comme point de repère. Aux étages supérieurs, il peut mentionner la sortie des ascenseurs ou des escaliers.

Un exemple complet d'instructions générées pourrait être :

> *Depuis l'entrée principale, dirigez-vous vers le centre du bâtiment où se trouvent les escaliers et l'ascenseur. Montez au 2e étage. À la sortie de l'ascenseur, tournez à gauche et suivez le couloir nord. Le laboratoire A-2110 se trouve à votre gauche après la deuxième intersection.*

Cette approche basée sur des gabarits offre un bon équilibre entre simplicité d'implémentation et qualité des instructions produites. Pour des applications futures, une amélioration possible consisterait à intégrer un module de génération de texte plus sophistiqué capable de varier les formulations et d'adapter le niveau de détail selon le profil de l'usager.

3.6.5 Interface utilisateur du prototype

Le prototype NaviUQTR a été implémenté sous forme d'application interactive dans un environnement Google Colab Notebook. Cette interface permet à l'usager de formuler des requêtes en langage naturel et d'obtenir des itinéraires avec instructions contextualisées.

L'interface comprend trois éléments principaux. Un encadré informatif présente le titre du système, une brève description des fonctionnalités, et un rappel que la modélisation est schématique à des fins de démonstration. Cet encadré inclut également l'inventaire complet des salles disponibles, organisées par étage (RDC au 4^{e} étage). Ces codes de salles sont cliquables, permettant à l'usager de sélectionner directement une destination sans la saisir manuellement. Des exemples de requêtes sont également affichés pour illustrer les formulations acceptées.

Une zone de saisie textuelle permet à l'usager de formuler librement sa requête (par exemple : « Itinéraire vers A-2110 », « Où est le laboratoire informatique ? »). L'usager peut soit taper sa requête, soit cliquer sur un code de salle dans la liste pour générer automatiquement la requête.

Deux boutons d'action complètent l'interface. Le bouton « Envoyer » déclenche le traitement complet : interprétation de la requête, identification par le LLM affiné, calcul du chemin via Dijkstra, génération d'instructions structurées. Le bouton « Nouvelle conversation » réinitialise l'interface pour une nouvelle requête.

L'interface a été développée avec la bibliothèque Python `ipywidgets`, facilitant le prototypage rapide. Les Figures 3.12 et 3.13 présentent cette interface.

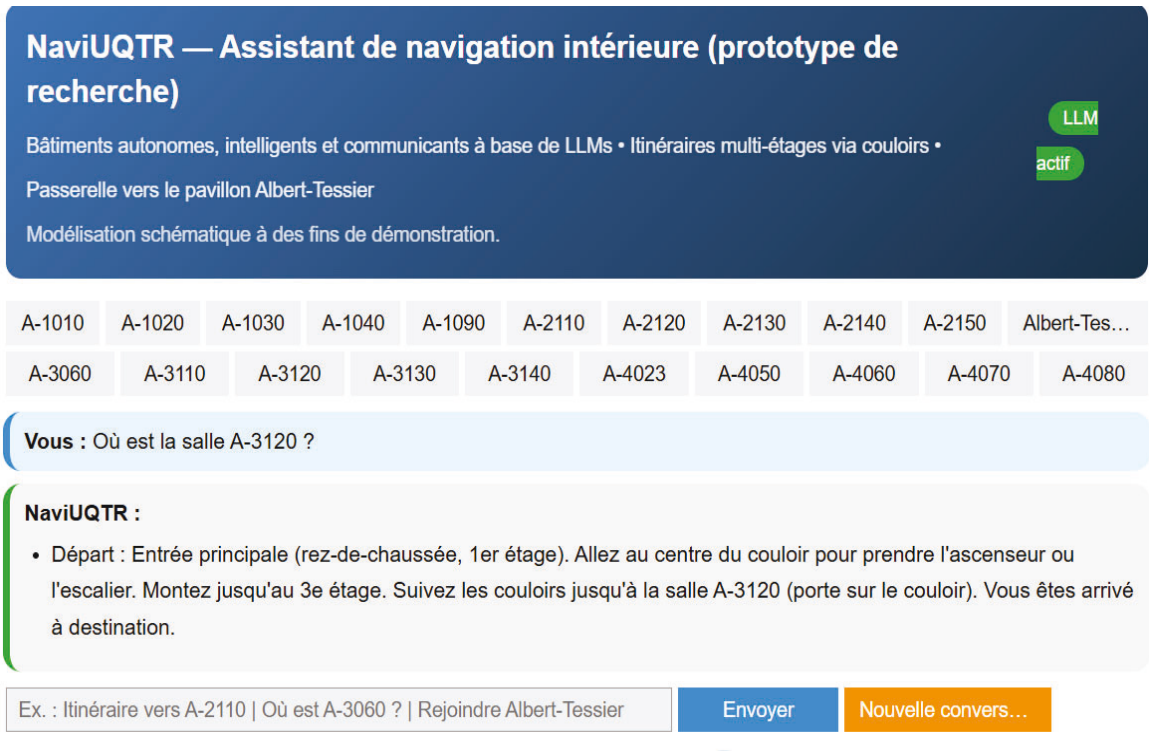

FIGURE 3.12 – Interface du prototype NaviUQTR développé avec Python (ipywidgets) — Partie 1 : Présentation du système, zone de saisie et résultats de navigation avec instructions détaillées..

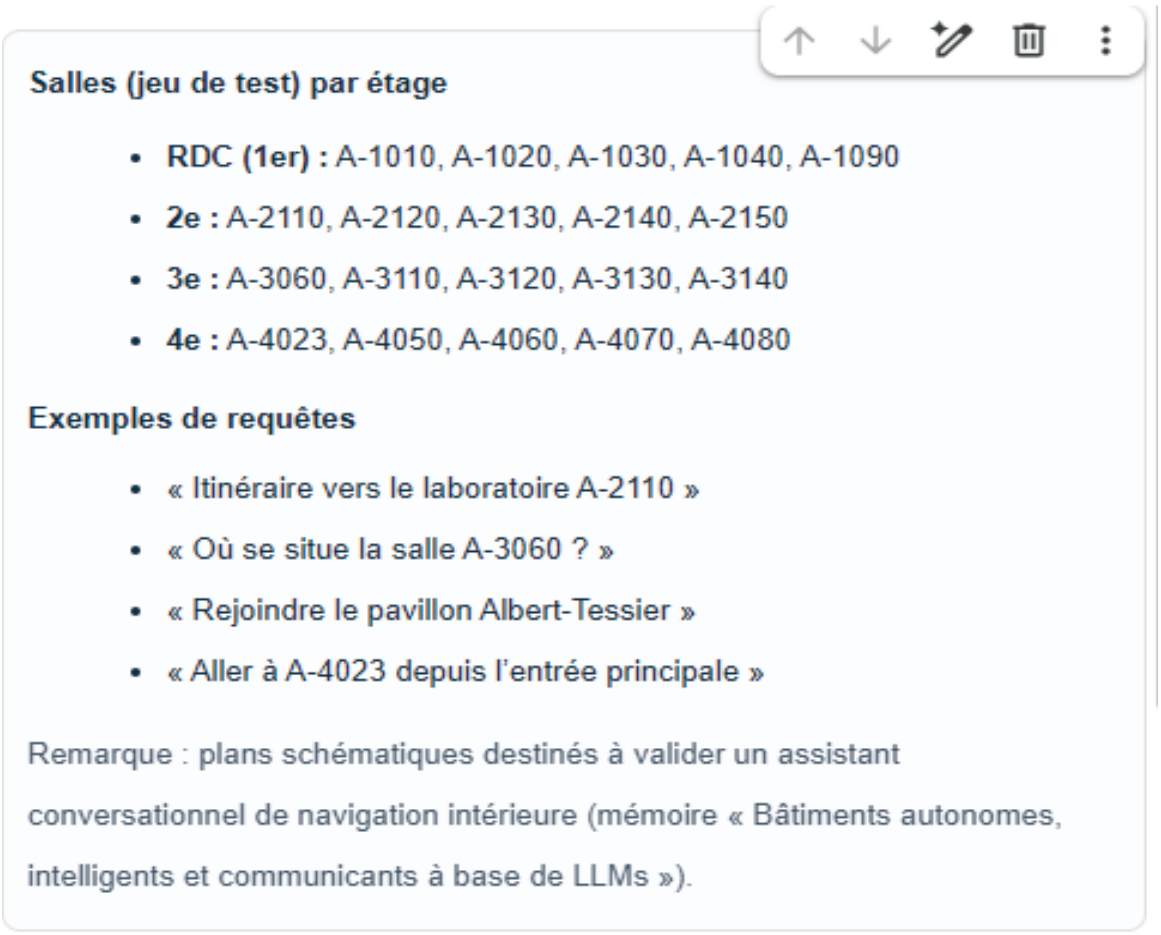

FIGURE 3.13 – Interface du prototype NaviUQTR — Partie 2 : liste des salles disponibles.

3.7 Métriques d'évaluation

L'évaluation du système NaviUQTR repose sur plusieurs métriques complémentaires permettant d'apprécier différentes dimensions de sa performance. Cette section présente les indicateurs retenus et justifie leur pertinence.

3.7.1 Précision d'identification de destination

La métrique principale évalue la capacité du modèle LLM affiné à identifier correctement la destination souhaitée à partir de requêtes variées. Pour chaque requête de test, le code de salle généré par le modèle est comparé au code attendu.

La précision est calculée simplement comme le rapport entre le nombre de prédictions correctes et le nombre total de requêtes :

$$\text{Précision} = \frac{\text{Nombre de prédictions correctes}}{\text{Nombre total de requêtes}} \times 100 \tag{3.1}$$

Un ensemble de test de 16 requêtes représentatives a été constitué, couvrant quatre catégories distinctes. La catégorie **identification directe** comprend 6 requêtes posant des questions du type « Où est A-2110 ? », visant à évaluer la capacité du système à traiter les codes de salles explicites. La catégorie **navigation classique** regroupe 5 requêtes formulées comme « Comment aller à A-3060 ? », testant la compréhension des intentions de déplacement. La catégorie ***alias* sémantiques** comporte 3 requêtes utilisant des descriptions fonctionnelles telles que « laboratoire informatique » ou « salle de réunion », évaluant la capacité du modèle à mapper des désignations naturelles vers des codes techniques. Enfin, la catégorie **passerelle** contient 2 requêtes concernant la connexion inter-bâtiments, testant la gestion des destinations spéciales du système.

Cette diversité permet d'évaluer séparément les performances sur chaque type de formulation, révélant ainsi les forces et faiblesses spécifiques de chaque version du modèle.

La Figure 3.14 présente la matrice de confusion obtenue pour la version finale V2.1.

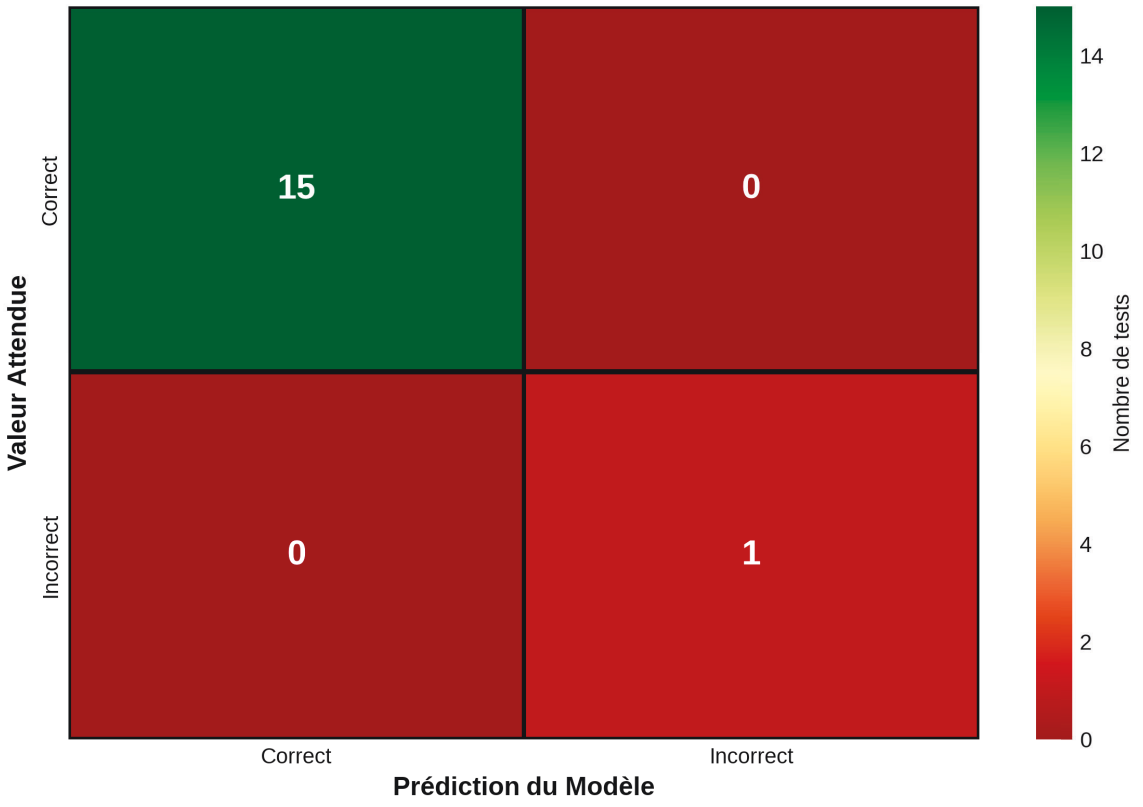

FIGURE 3.14 – Matrice de confusion du système NaviUQTR V2.1. Sur 16 tests, 15 prédictions sont correctes (93,8% de précision).

3.7.2 Analyse des métriques d'entraînement

Bien que la précision sur l'ensemble de test constitue l'indicateur principal, une attention particulière a également été portée aux métriques d'entraînement, notamment le ratio validation/entraînement. Ce ratio, calculé comme le quotient de la perte de validation par la perte d'entraînement, fournit un indicateur de la capacité de généralisation du modèle.

Le ratio validation/entraînement est utilisé ici comme indicateur relatif de l'équilibre entre spécialisation et généralisation, et non comme mesure absolue de performance. Il convient de préciser que ce ratio ne constitue pas en lui-même une garantie de qualité : un ratio proche de 1,0 peut demeurer insatisfaisant si les deux pertes sont absolument élevées, indiquant que le modèle n'a pas convergé vers une solution acceptable. Dans notre cas, les valeurs absolues des pertes d'entraînement (0,092 à 0,258) et de validation (0,135 à 0,284) demeurent dans des plages raisonnables pour une tâche d'instruction-tuning sur corpus spécialisé, ce qui rend le ratio pertinent comme indicateur comparatif entre versions. Un ratio proche de 1,0 suggère alors que le modèle généralise aussi bien sur des données non vues que sur les données d'entraînement, tandis qu'un ratio significativement supérieur à 1,5 indique une spécialisation excessive sur les patterns d'entraînement. Cette utilisation du ratio comme métrique comparative entre versions s'inspire des pratiques documentées en apprentissage supervisé pour l'évaluation du surapprentissage [34], [42].

Comme présenté dans le Tableau 3.2, V2.1 présente le meilleur ratio (1,10), confirmant l'efficacité de la stratégie d'équilibrage du corpus.

3.7.3 Temps de réponse

Bien que la précision constitue l'objectif principal, la réactivité du système est également importante pour l'expérience utilisateur. Le temps de réponse global a été mesuré en le décomposant en plusieurs phases distinctes. Le temps d'interprétation, correspondant à l'analyse par *pattern matching*, est inférieur à 10 ms et donc négligeable. Le temps d'inférence LLM, phase la plus coûteuse, varie entre 200 et 500 ms selon la longueur de la requête. Le temps de calcul d'itinéraire via l'algorithme de Dijkstra reste inférieur à 10 ms pour le graphe actuel. Enfin, le temps de génération d'instructions, transformant le chemin en consignes verbales, demeure sous la barre des 50 ms.

Le temps de réponse total moyen se situe entre 300 et 600 millisecondes, largement acceptable pour une utilisation interactive. L'étape la plus coûteuse est l'inférence

LLM, mais avec l'utilisation de LoRA et de la quantification FP16, ce temps reste raisonnable même sur GPU standard.

3.8 Environnement technique

Le système NaviUQTR a été développé dans un environnement technique combinant plusieurs technologies complémentaires. Cette section détaille les choix technologiques et l'infrastructure utilisée.

3.8.1 Infrastructure logicielle

Le langage de programmation retenu est Python (version 3.10), en raison de sa richesse écosystémique pour le traitement du langage naturel et l'analyse de graphes. Les bibliothèques principales utilisées incluent **Transformers** (version 4.36.0), bibliothèque Hugging Face pour le chargement et l'affinage des modèles de langage. **PEFT** (version 0.7.0) fournit l'implémentation de la technique LoRA pour l'affinage efficace. **NetworkX** (version 3.1) assure la manipulation du graphe topologique et l'implémentation de l'algorithme de Dijkstra. **Datasets** (version 2.15.0) gère les corpus d'entraînement au format JSONL (*JavaScript Object Notation Lines*). L'interface utilisateur interactive repose sur **ipywidgets** (version 8.1.0) dans l'environnement Google Colab. Enfin, **Matplotlib** (version 3.8.0) permet la visualisation des plans d'étage et la génération de figures.

La manipulation du graphe topologique repose sur NetworkX, qui fournit des implémentations efficaces des algorithmes de recherche de chemin. Les attributs des nœuds et des arêtes sont stockés directement dans les structures de données de NetworkX, facilitant ainsi leur manipulation.

3.8.2 Infrastructure matérielle

L'entraînement des modèles a été effectué sur Google Colab avec accès à un processeur graphique (GPU, *Graphics Processing Unit*) NVIDIA A100-SXM4-80GB. Cette configuration offre plusieurs avantages décisifs. La mémoire GPU de 80 GB s'avère suffisante pour l'affinage avec LoRA, même pour les corpus les plus volumineux. Le support de la précision mixte FP16 accélère significativement l'entraînement tout en préservant la qualité des gradients. Les temps d'entraînement varient entre 1h et 4h selon la taille du corpus, durée acceptable pour une démarche itérative d'expérimentation. L'utilisation de Colab présente l'avantage de la reproductibilité : tout chercheur disposant d'un compte Google peut reproduire les expériences sans investissement matériel.

3.8.3 Organisation du code

L'ensemble du code est organisé en modules distincts suivant une architecture claire et modulaire. Le module **GraphBuilder** prend en charge la construction du graphe topologique multi-niveaux et l'enrichissement sémantique des nœuds. **CorpusGenerator** gère la génération synthétique des corpus d'entraînement dans leurs différentes versions. **FineTuner** encapsule toute la logique d'affinage avec LoRA et Transformers, incluant la gestion des hyperparamètres et du processus d'entraînement.**NavigationAgent** implémente le pipeline complet de navigation, de l'interprétation des requêtes à la génération d'instructions en passant par l'identification LLM et le calcul de chemin avec Dijkstra. Enfin, le module **Interface** fournit l'interface utilisateur interactive développée avec ipywidgets.

Cette architecture modulaire facilite la maintenance et les évolutions futures du système. Chaque module peut être testé et amélioré indépendamment.

3.9 Conclusion

Ce chapitre a présenté en détail la méthodologie développée pour concevoir NaviU-QTR, un système de navigation intérieure intelligent. L'approche proposée repose sur quatre piliers complémentaires, articulant représentation spatiale, génération de corpus, affinage itératif et traitement du langage naturel.

La modélisation topologique sous forme de graphe multi-niveaux offre une abstraction à la fois précise et computationnellement efficace. Elle capture les éléments essentiels de la structure du modèle (couloirs, intersections, salles, équipements verticaux) tout en facilitant l'application d'algorithmes de recherche de chemin. L'approche schématique adoptée permet de valider les concepts méthodologiques sans les complications d'une architecture complexe, tout en préservant la transposabilité vers des environnements réels.

La constitution du corpus d'entraînement, à travers une démarche itérative V1 $\rightarrow$ V2 $\rightarrow$ V2.1, constitue l'un des apports méthodologiques majeurs de ce travail. Plutôt que de considérer le corpus comme une donnée figée, l'importance d'un processus d'amélioration progressive guidé par l'analyse des résultats a été démontrée. La progression de 37,5% (V1) à 75,0% (V2) puis 93,8% (V2.1) illustre l'efficacité de cette approche empirique.

L'adaptation du modèle Mistral-7B par affinage avec LoRA représente le cœur technique du système. La configuration retenue ($r=64$, $\alpha=128$) offre un excellent compromis entre efficacité paramétrique (2,26% des paramètres entraînables) et perfor-

mances. Le ratio validation/entraînement de 1,10 obtenu en V2.1 témoigne d'un équilibre optimal entre spécialisation sur le domaine cible et capacité de généralisation.

Le pipeline complet, articulant interprétation des requêtes, identification par LLM, calcul d'itinéraires via Dijkstra et génération d'instructions, offre une expérience utilisateur fluide et naturelle. La modularité de l'architecture facilite les évolutions futures et l'intégration de nouvelles fonctionnalités (guidage vocal, mise à jour en temps réel, intégration de capteurs de localisation).

L'environnement technique choisi, centré sur Python et l'écosystème Hugging Face, offre un bon compromis entre facilité de développement et possibilités d'évolution. Bien qu'initialement conçu comme un prototype de recherche, le système possède les fondations nécessaires pour évoluer vers une application opérationnelle.

Le chapitre suivant présentera les résultats expérimentaux obtenus avec NaviUQTR. Les performances du système selon les différentes métriques définies y seront analysées, l'approche proposée sera comparée aux solutions existantes, et les enseignements tirés de cette première mise en œuvre seront discutés. Cette évaluation permettra de valider les choix méthodologiques effectués et d'orienter les développements futurs du système.

Chapitre 4 – Résultats et discussion

4.1 Introduction

Ce chapitre présente les résultats expérimentaux obtenus avec le système NaviUQTR et en propose une analyse critique. Une démarche itérative d'amélioration progressive a été adoptée, développant trois versions successives du système (V1, V2, V2.1) dont les performances sont comparées. L'évaluation repose sur les métriques définies au chapitre précédent : précision d'identification de destination, qualité de l'affinage, et performance du calcul d'itinéraires.

Les résultats de chaque version sont d'abord analysés, en identifiant leurs forces et limitations respectives. Une analyse comparative des trois versions est ensuite présentée, mettant en évidence les enseignements méthodologiques tirés de cette progression itérative. NaviUQTR est ensuite comparé aux approches existantes, tant traditionnelles que basées sur l'apprentissage profond. Les performances techniques du système sont enfin examinées avant de discuter des limitations identifiées et des perspectives d'amélioration.

L'ensemble des évaluations repose sur un corpus de test de 16 requêtes représentatives, couvrant quatre catégories distinctes. La catégorie identification directe comprend 6 requêtes du type « Où est A-2110 ? », testant la reconnaissance des codes explicites. La catégorie navigation classique regroupe 5 requêtes formulées comme « Comment aller à A-3060 ? », évaluant la compréhension des intentions de déplacement. La catégorie *alias* sémantiques comporte 3 requêtes utilisant des descriptions fonctionnelles (« laboratoire informatique », « salle de réunion »), testant la capacité du modèle à mapper des désignations naturelles vers des codes techniques. Enfin, la catégorie passerelle contient 2 requêtes concernant la connexion inter-bâtiments. Cette diversité permet d'évaluer séparément les performances sur chaque type de formulation, révélant ainsi les forces et faiblesses spécifiques de chaque version.

4.1.1 Stabilité et reproductibilité des résultats

Avant de présenter les résultats détaillés de chaque version, il convient de démontrer la fiabilité et la reproductibilité des mesures obtenues. Une limite inhérente à l'évaluation proposée réside dans la taille restreinte de l'ensemble de test (16 requêtes). Afin de répondre à cette préoccupation, une analyse de stabilité a été conduite en répétant l'inférence cinq fois sur le même ensemble de test pour chaque version du système.

Les paramètres de génération quasi-déterministes adoptés (température $T = 0{,}05$, *top-p* $= 0{,}95$) minimisent théoriquement la variabilité entre répétitions. À cette température, le modèle sélectionne quasi-systématiquement le *token* de plus haute probabilité, produisant des sorties pratiquement identiques pour une même entrée. Le Tableau 4.1 présente les résultats obtenus.

TABLE 4.1 – Analyse de stabilité : précision globale sur 5 répétitions d'inférence pour chaque version (n=16 requêtes par répétition)

Version	**R1**	**R2**	**R3**	**R4**	**R5**	**Écart-type**
V1	37,5%	37,5%	37,5%	37,5%	37,5%	0,0%
V2	75,0%	75,0%	75,0%	75,0%	75,0%	0,0%
V2.1	93,8%	93,8%	93,8%	93,8%	93,8%	0,0%

Les résultats sont parfaitement stables à travers les cinq répétitions pour les trois versions, avec un écart-type nul. Cette stabilité confirme que les mesures de précision présentées dans les sections suivantes sont fiables et reproductibles, malgré la taille limitée de l'ensemble de test.

Il convient néanmoins de reconnaître que cette stabilité ne compense pas entièrement la taille restreinte de l'ensemble de test. Une évaluation sur un corpus plus large, idéalement constitué de requêtes réelles d'utilisateurs, fournirait une mesure de généralisation plus robuste et constitue une priorité identifiée pour les travaux futurs.

4.2 Version V1 : Phase exploratoire

4.2.1 Caractéristiques du corpus V1

La première version du système repose sur un corpus d'entraînement de 1 624 exemples générés synthétiquement. Comme détaillé à la section 3.4, lors de cette phase initiale, aucune attention particulière n'avait été accordée à l'équilibre entre les différents types de questions. L'analyse a posteriori révèle une distribution fortement déséquilibrée où environ 60% des questions utilisaient la formulation « Comment aller à... ? », seulement 9% employaient « Où est... ? », et moins de 5% contenaient des *alias* sémantiques. Cette composition biaisée a eu des conséquences directes sur les performances du modèle.

L'affinage de Mistral-7B avec la configuration LoRA (r=64, α=128) décrite à la section 3.5 s'est effectué en 5 époques sur le processeur graphique (GPU) A100, pour une durée totale de 59 minutes. Les métriques d'entraînement montrent une perte d'entraînement de 0,092 et une perte de validation de 0,135, soit un ratio de 1,47 indiquant un léger surapprentissage.

4.2.2 Résultats V1

Le Tableau 4.2 présente les performances obtenues sur l'ensemble de test.

TABLE 4.2 – Résultats de la version V1 sur le corpus de test (n=16 requêtes)

Catégorie	**Réussite**	**Précision**
Identification directe (6 requêtes)	0/6	0%
Navigation classique (5 requêtes)	5/5	100%
alias sémantiques (3 requêtes)	3/3	100%
Passerelle (2 requêtes)	0/2	0%
Total	**6/16**	**37,5%**

Ces résultats révèlent un déséquilibre majeur dans les capacités du modèle. D'une part, le système réussit parfaitement les questions de navigation classique (100%), catégorie sur-représentée dans le corpus d'entraînement. Le modèle a également appris à traiter correctement les trois *alias* sémantiques testés, bien que cette performance soit trompeuse compte tenu du très faible nombre d'exemples disponibles.

D'autre part, l'échec complet sur les questions d'identification directe (0%) constitue une limitation critique. Face à une question du type « Où est A-2110 ? », le modèle génère systématiquement des réponses inappropriées, ayant appris à associer toute requête à une intention de navigation. Cette confusion s'explique directement par la sur-représentation des formulations « Comment aller » dans le corpus d'entraînement.

4.2.3 Analyse des limitations V1

L'analyse approfondie révèle que le problème ne réside pas dans la capacité du modèle à apprendre, mais dans la composition déséquilibrée du corpus. Le modèle a effectivement appris les patterns présents dans les données d'entraînement, mais ces patterns ne reflètent pas la diversité des requêtes réelles. Cette observation oriente directement la stratégie d'amélioration pour V2 : augmenter massivement le corpus en assurant une meilleure représentation de tous les types de questions, particulièrement les formulations d'identification directe sous-représentées en V1.

La performance surprenante sur les *alias* (100%) mérite également une analyse nuancée. Cette réussite repose sur seulement 3 exemples de test issus d'un corpus d'entraînement lui-même pauvre en *alias*, soulevant des questions légitimes sur la capacité de généralisation réelle du modèle. Cette fragilité deviendra évidente lors de l'évaluation de V2.

4.3 Version V2 : Optimisation par augmentation

4.3.1 Stratégie d'augmentation du corpus

Face aux limitations de V1, une seconde version privilégiant une augmentation substantielle du nombre d'exemples a été développée, comme décrit à la section 3.4. L'hypothèse formulée était qu'un corpus plus large permettrait au modèle de mieux généraliser et de corriger les biais observés.

La génération de V2 a suivi un processus d'énumération systématique et combinatoire. Cette approche a produit un corpus de 13 152 exemples d'entraînement (9 206 pour l'entraînement, 3 946 pour la validation), soit une augmentation d'un facteur 8 par rapport à V1. L'entraînement s'est effectué en 5 époques sur GPU A100 pour une durée totale de 3h53. Les métriques d'entraînement montrent une perte d'entraînement de 0,186 et une perte de validation de 0,315, soit un ratio de 1,69 indiquant un surapprentissage plus marqué qu'en V1.

4.3.2 Résultats V2

Le Tableau 4.3 présente les performances de la version V2.

TABLE 4.3 – Résultats de la version V2 sur le corpus de test (n=16 requêtes)

Catégorie	**Réussite**	**Précision**
Identification directe (6 requêtes)	6/6	100%
Navigation classique (5 requêtes)	5/5	100%
alias sémantiques (3 requêtes)	0/3	0%
Passerelle (2 requêtes)	1/2	50%
Total	**12/16**	**75,0%**

L'évaluation de V2 révèle une amélioration significative de la précision globale, passant de 37,5% à 75,0%. Le problème d'identification directe a été complètement résolu, le modèle atteignant désormais 100% de réussite sur les questions « Où est A-2110 ? ». La navigation classique maintient également sa performance parfaite à 100%.

Cependant, un nouveau problème critique est apparu. La précision sur les *alias* sémantiques a chuté à 0%, alors qu'elle était à 100% en V1. Face à une requête comme « Où est le laboratoire informatique ? », le modèle échoue systématiquement à identifier la destination A-2110. Cette régression paradoxale a conduit à analyser en profondeur la composition du corpus V2.

4.3.3 Analyse du phénomène de dilution

L'analyse révèle que malgré l'augmentation massive du nombre total d'exemples, la proportion d'*alias* sémantiques est restée très faible dans le corpus V2 (moins de 5%). Le phénomène de dilution a joué en défaveur de cette catégorie. Les exemples d'*alias*, bien qu'en nombre absolu légèrement supérieur à V1, sont noyés dans la masse des exemples utilisant des codes de salles explicites.

Ce constat illustre un principe important en apprentissage automatique découvert empiriquement : l'augmentation quantitative du corpus ne garantit pas automatiquement une amélioration qualitative si la composition relative des différentes catégories n'est pas soigneusement contrôlée. Le modèle a appris à privilégier les patterns majoritaires (codes explicites) au détriment des patterns minoritaires (*alias*), même lorsque ces derniers sont essentiels pour l'utilisabilité du système.

Cette analyse oriente directement la stratégie pour V2.1. Plutôt que de continuer à augmenter la taille du corpus, l'équilibrage qualitatif de sa composition doit être privilégié, en garantissant une représentation substantielle de chaque catégorie de questions, particulièrement les *alias* sémantiques.

4.4 Version V2.1 : Équilibrage qualitatif

4.4.1 Stratégie d'équilibrage du corpus

Pour la troisième version, une approche radicalement différente a été adoptée en privilégiant l'équilibrage qualitatif plutôt que la simple augmentation quantitative, comme détaillé à la section 3.4. Plutôt que d'augmenter encore le nombre total d'exemples, la taille du corpus a été réduite tout en garantissant une représentation équilibrée de chaque catégorie de questions.

La composition de V2.1 repose sur quatre principes fondamentaux. D'abord, une représentation équilibrée des *alias* où 40% des exemples utilisent des formulations avec *alias* sémantiques. Ensuite, le maintien de codes explicites avec 60% des exemples conservant l'utilisation directe de codes de salles. Par ailleurs, la diversité syntaxique est préservée grâce au maintien de la variété des formulations. Enfin, la taille a été optimisée pour produire un corpus final de 3 942 exemples (2 768 entraînement, 1 174 validation).

L'entraînement s'est effectué en 5 époques sur le processeur graphique (GPU) A100 pour une durée de 1h06. Les métriques d'entraînement montrent une perte d'entraînement de 0,258 et une perte de validation de 0,284, soit un ratio de 1,10. Ce ratio,

significativement meilleur que ceux de V1 (1,47) et V2 (1,69), témoigne d'un équilibre optimal entre spécialisation sur le domaine cible et capacité de généralisation.

4.4.2 Résultats V2.1

Le Tableau 4.4 présente les performances de la version finale.

TABLE 4.4 – Résultats de la version V2.1 sur le corpus de test (n=16 requêtes)

Catégorie	**Réussite**	**Précision**
Identification directe (6 requêtes)	6/6	100%
Navigation classique (5 requêtes)	5/5	100%
alias sémantiques (3 requêtes)	3/3	100%
Passerelle (2 requêtes)	1/2	50%
Total	**15/16**	**93,8%**

L'évaluation de V2.1 confirme le bien-fondé de l'approche d'équilibrage qualitatif. La précision globale atteint 93,8%, soit une amélioration de 18,8 points par rapport à V2 et de 56,2 points par rapport à V1. Plus significativement, toutes les catégories de questions atteignent désormais des performances excellentes. L'identification directe atteint 100% de réussite, performance maintenue depuis V2. La navigation classique conserve également 100% de réussite, stabilité observée depuis V1. Les *alias* sémantiques, problème majeur de V2, sont désormais résolus avec 100% de réussite. Seule la catégorie passerelle présente une faiblesse résiduelle à 50%, dont l'impact demeure cependant limité.

4.5 Analyse comparative des trois versions

4.5.1 Synthèse des résultats

Le Tableau 4.5 synthétise l'ensemble des résultats obtenus.

TABLE 4.5 – Synthèse comparative des trois versions de NaviUQTR

Caractéristique	**V1**	**V2**	**V2.1**
Taille corpus	1 624	13 152	3 942
Composition	Déséquilibrée	Codes explicites	Équilibrée
alias (%)	$< 5\%$	$< 5\%$	40%
Ratio val/train	1,47	1,69	**1,10**
Précision globale	37,5%	75,0%	**93,8%**
Identification	0%	100%	100%
Navigation	100%	100%	100%
alias	100%*	0%	100%
Passerelle	0%	50%	50%

* Petit corpus, généralisation douteuse

La progression des performances à travers les trois versions, illustrée précédemment à la Figure 3.7, démontre l'efficacité de l'approche itérative d'amélioration. L'évolution par catégorie de requêtes, présentée à la Figure 3.8, révèle comment chaque version a corrigé les faiblesses de la précédente tout en préservant ses forces.

4.5.2 Enseignements méthodologiques

Cette progression itérative V1→V2→V2.1 démontre plusieurs principes importants en apprentissage automatique découverts empiriquement.

Premièrement, l'augmentation quantitative du corpus ne garantit pas automatiquement une amélioration qualitative. V2, avec un corpus 8 fois plus grand que V1, améliore certaines performances (identification directe) mais dégrade d'autres (*alias*). Cette observation contredit l'intuition naïve selon laquelle « plus de données » équivaut nécessairement à « meilleur modèle ».

Deuxièmement, la composition relative des catégories s'avère plus déterminante que le nombre absolu d'exemples. V2.1, avec un corpus plus petit que V2 (3 942 vs 13 152), atteint des performances supérieures (93,8% vs 75,0%) grâce à un équilibrage qualitatif. Le ratio 40%/60% entre *alias* et codes explicites s'est révélé optimal pour ce cas d'usage.

Troisièmement, le ratio validation/entraînement constitue un indicateur fiable de la capacité de généralisation. V2.1 présente le meilleur ratio (1,10), confirmant que l'équilibrage du corpus réduit le surapprentissage et améliore la robustesse. Ce ratio constitue désormais une métrique précoce pour évaluer la qualité d'un corpus avant même de mesurer la précision finale.

Quatrièmement, une démarche itérative guidée par l'analyse des erreurs s'avère plus efficace qu'une optimisation directe. Le problème de dilution des *alias* n'aurait probablement pas été identifié sans l'expérience de V2. Cette approche empirique, bien que plus longue, permet de comprendre en profondeur les mécanismes d'apprentissage et d'éviter les solutions superficielles.

La Figure 4.1 illustre la relation non-linéaire entre taille du corpus et performance.

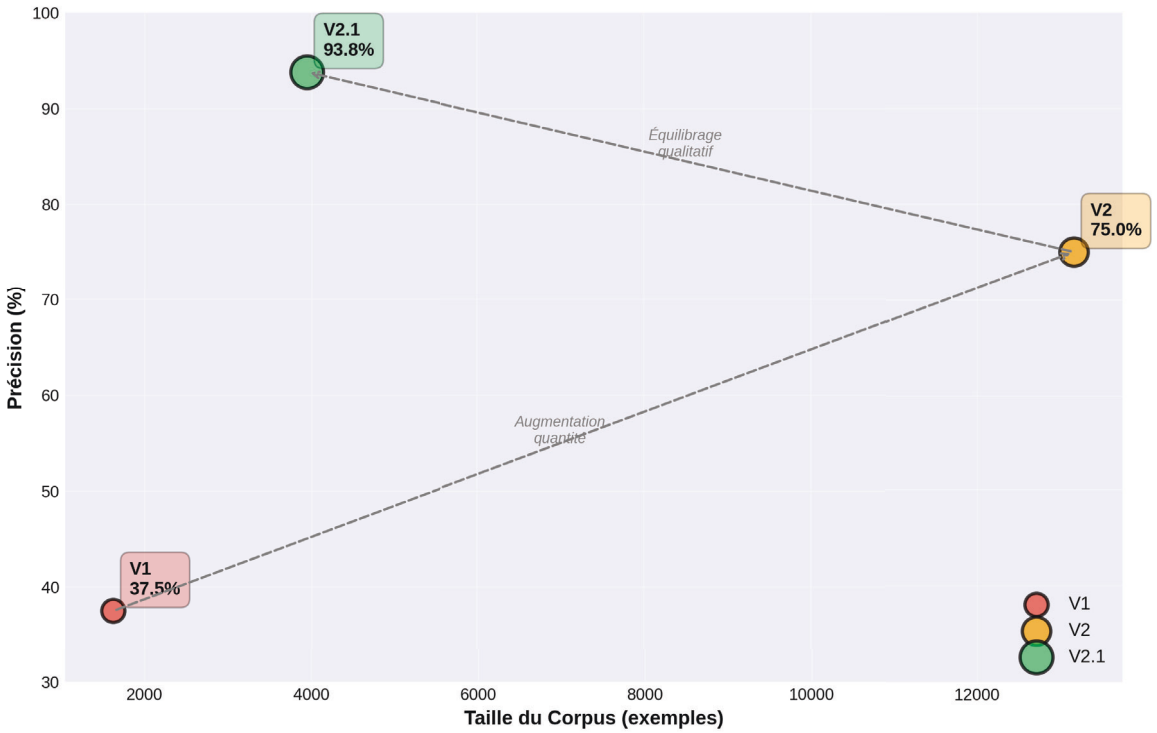

FIGURE 4.1 – Relation entre taille du corpus et performance. La version V2.1 démontre qu'un corpus plus petit mais mieux équilibré surpasse un corpus massif mais déséquilibré.

4.6 Comparaison avec les approches existantes

4.6.1 Positionnement par rapport aux systèmes traditionnels

Le Tableau 4.6 compare NaviUQTR aux approches traditionnelles de navigation intérieure les plus représentatives. Cette comparaison n'est pas exhaustive : l'état de l'art dans ce domaine évolue rapidement et certains systèmes récents n'ont pu être intégrés, soit parce que leurs données d'évaluation ne sont pas publiquement disponibles, soit parce que leurs protocoles d'évaluation diffèrent significativement de celui adopté dans ce travail.

TABLE 4.6 – Comparaison de NaviUQTR avec les systèmes traditionnels de navigation intérieure

Caractéristique	**Plans statiques**	**Applications GPS intérieur**	**NaviUQTR V2.1**
Langage naturel	Non	Limitée	Oui
Personnalisation	Non	Faible	Élevée
Infrastructure	Aucune	Balises/WiFi	Aucune
Coût déploiement	Très faible	Élevé	Faible
Accessibilité PMR	Non	Variable	Oui
Interface vocale	Non	Variable	Non
Support BIM	Non	Partiel	Non
Précision	–	85–90%	93,8%

NaviUQTR se distingue des plans statiques par sa capacité à interpréter des requêtes en langage naturel et à générer des itinéraires personnalisés. Contrairement aux applications de positionnement intérieur, il ne requiert aucune infrastructure de localisation coûteuse telle que des balises Bluetooth ou des points d'accès WiFi calibrés. La personnalisation constitue un avantage majeur, permettant la prise en compte des contraintes d'accessibilité et l'adaptation aux préférences individuelles. Le coût de déploiement demeure modéré, évitant les investissements matériels importants requis par les systèmes de positionnement intérieur. Une interface de programmation applicative (API) LLM externe peut être utilisée de manière optionnelle, sans constituer une dépendance infrastructurelle obligatoire.

Deux fonctionnalités absentes de la version actuelle méritent d'être mentionnées : l'interface vocale et le support des modèles BIM (*Building Information Modeling*). Ces capacités permettraient aux visiteurs d'interagir directement dans leur langue sans saisie textuelle, et d'enrichir le graphe topologique avec des données architecturales réelles. Leur intégration est identifiée comme priorité à court terme dans les perspectives de développement du système.

Il convient enfin de reconnaître que la comparaison présentée se concentre sur les caractéristiques qualitatives les plus pertinentes pour l'usage ciblé — la navigation humaine autonome dans un bâtiment universitaire — plutôt que sur une mise en concurrence numérique directe, dans un souci de rigueur méthodologique plutôt que d'exhaustivité.

4.6.2 Comparaison avec les LLMs pour la navigation

Le Tableau 4.7 positionne NaviUQTR par rapport aux approches récentes exploitant les modèles de langage pour la navigation et la compréhension spatiale. Cette comparaison est organisée en trois catégories reflétant les différents paradigmes d'application des LLMs dans le domaine spatial.

TABLE 4.7 – Comparaison des approches à base de modèles de langage pour la navigation et la compréhension spatiale

Système (Année)	Modèle	Domaine	Performance	Contribution principale
Navigation intérieure avec modèles de langage				
Zhang et al. [3] (2024)	GPT (accès API)	Navigation intérieure (multi-graphes)	88–92%	Fusion multi-graphes et modèle de langage
NaviUQTR V2.1 (2026)	**Mistral-7B + LoRA**	**Navigation intérieure (graphe topologique)**	**93,8%**	**Affinage itératif sur corpus équilibré**
Navigation par agents incarnés et vision-langage (VLN)				
MapGPT [2] (2024)	GPT-4 (*zero-shot*)	Navigation VLN robotique	85–90% (taux de succès)	Carte linguistique construite en ligne
PaLM-E [43] (2023)	PaLM-E 562 milliards	Robotique multimodale	87%	Fusion vision, langage et action
Autres applications spatiales des modèles de langage				
PathGen-LLM [1] (2025)	Modèle de langage affiné	Transport urbain dynamique	∼89%	Génération d'itinéraires en temps réel

Note : Les métriques de performance varient selon les environnements et les protocoles d'évaluation propres à chaque système. Une comparaison directe des valeurs numériques doit être interprétée avec prudence.

Navigation intérieure : Dans cette catégorie, NaviUQTR V2.1 obtient une précision supérieure (93,8%) à Zhang et al. [3] (88-92%), grâce à l'hybridation entre planification algorithmique (Dijkstra) et compréhension linguistique (LLM affiné). Cette approche hybride s'avère déterminante : les systèmes entièrement neuronaux, bien qu'impressionnants, souffrent encore d'erreurs de raisonnement spatial conduisant à des itinéraires sous-optimaux ou invalides. En déléguant le calcul de chemins à un algorithme éprouvé garantissant l'optimalité mathématique, ces écueils sont évités tout en bénéficiant de la flexibilité linguistique du LLM pour l'interprétation des requêtes. De plus, le choix d'un modèle source ouverte (Mistral-7B) affiné avec LoRA garantit transparence et reproductibilité, contrairement aux solutions basées sur des API propriétaires.

Navigation embodied et VLN : MapGPT [2] et PaLM-E [43] représentent l'état de l'art en navigation robotique embodied. MapGPT construit dynamiquement une

carte linguistique pour guider un agent GPT-4 en mode zero-shot, atteignant 85 à 90% de taux de succès sur les références comparatives (*benchmarks*) R2R et REVERIE. PaLM-E, avec ses 562 milliards de paramètres, intègre vision, langage et contrôle moteur pour la robotique multimodale. Bien que ces systèmes traitent des entrées visuelles là où NaviUQTR ne traite que du langage, ils partagent l'approche fondamentale d'utiliser des représentations spatiales explicites (cartes, graphes) pour structurer le raisonnement du modèle. Cette convergence méthodologique valide l'importance des structures topologiques pour guider efficacement les LLMs dans les tâches spatiales.

Autres applications spatiales : PathGen-LLM [1] illustre l'applicabilité des modèles de langage à d'autres domaines connexes. Ce système génère des itinéraires dynamiques dans les réseaux de transport urbain en intégrant des données de trafic en temps réel, confirmant la tendance générale vers l'hybridation entre structures algorithmiques classiques et capacités de compréhension des modèles de langage pour résoudre des problèmes spatiaux complexes.

La contribution distinctive de NaviUQTR réside dans trois choix méthodologiques complémentaires. Premièrement, l'utilisation d'un modèle à code source ouvert (Mistral-7B, licence Apache 2.0) facilitant la reproductibilité scientifique. Deuxièmement, la stratégie d'optimisation itérative du corpus d'entraînement (V1→V2→V2.1) ayant permis d'identifier empiriquement l'importance critique de l'équilibrage qualitatif sur l'augmentation quantitative. Troisièmement, l'hybridation explicite entre planification algorithmique garantie (Dijkstra) et compréhension linguistique flexible (modèle de langage affiné), évitant ainsi les erreurs de raisonnement spatial observées dans les approches purement neuronales.

4.6.3 Justification du choix de Mistral-7B

Le Tableau 4.8 compare Mistral-7B aux alternatives considérées lors de la phase de sélection du modèle de base.

TABLE 4.8 – Comparaison des modèles de langage évalués

Modèle	Taille	Accès	Affinage LoRA	Performance	Latence
GPT-4	∼1 760B	API commerciale	Non	Excellente	Élevée
GPT-3.5	∼175B	API commerciale	Non	Bonne	Moyenne
LLaMA-2 7B	7B	Poids accessibles	Oui	Bonne	Moyenne
LLaMA-2 13B	13B	Poids accessibles	Oui	Très bonne	Élevée
Mistral-7B	**7B**	**Apache 2.0**	**Oui**	**Très bonne**	**Faible**

Il convient de préciser les distinctions terminologiques relatives à l'accessibilité des modèles présentés dans ce tableau. LLaMA-2 7B et LLaMA-2 13B sont accessibles

sous licence restreinte réservée à l'usage de recherche uniquement, la redistribution commerciale étant soumise à conditions. Mistral-7B est distribué sous licence Apache 2.0, qui autorise explicitement l'utilisation commerciale, la modification et la redistribution, avec obligation de mention de la licence d'origine. Les poids du modèle sont librement téléchargeables sur la plateforme Hugging Face. Cette licence est l'une des plus permissives du domaine et constitue la définition communément admise d'un modèle « à source ouverte » dans la communauté de recherche en apprentissage automatique.

Plus généralement, un modèle dit « à code source ouvert » (*source ouverte*) au sens strict désigne un modèle dont le code d'entraînement, les données d'entraînement et les poids sont tous publiquement accessibles et librement redistribuables. Dans la pratique, cette définition stricte est rarement satisfaite par les modèles de grande taille. Le terme est couramment utilisé dans la communauté pour désigner des modèles dont les poids sont accessibles sous des licences permissives. GPT-4 et GPT-3.5, accessibles uniquement via interface de programmation applicative (API, *Application Programming Interface*) commerciale sans accès aux poids ni au code d'entraînement, sont qualifiés de modèles « propriétaires » (*closed-source*) au sens où leur architecture interne et leurs paramètres ne sont pas divulgués publiquement.

Une question légitime concerne la pertinence d'utiliser un modèle de langage de 7 milliards de paramètres pour une tâche de navigation dans un environnement comportant moins de 20 destinations. En apparence, cette disproportion entre la capacité du modèle et la complexité de la tâche pourrait sembler excessive. Plusieurs arguments justifient néanmoins ce choix.

Premièrement, l'objectif du présent travail dépasse la navigation dans un seul bâtiment : il s'agit de démontrer la faisabilité et la méthodologie d'une approche hybride LLM-graphe applicable à tout environnement intérieur, indépendamment de sa taille. Un modèle plus petit (1B à 3B paramètres) pourrait suffire pour 20 destinations, mais sa capacité de généralisation à des formulations non vues serait significativement réduite, compromettant la robustesse de l'approche sur des bâtiments réels plus complexes.

Deuxièmement, le choix de Mistral-7B est motivé par ses capacités de raisonnement spatial documentées [37] et sa compatibilité native avec LoRA, qui ramène le nombre de paramètres effectivement entraînés à seulement 157 millions (2,26%), réduisant ainsi la disproportion computationnelle.

Troisièmement, la question de la scalabilité est précisément l'une des perspectives identifiées : transposer NaviUQTR à un bâtiment réel comportant des centaines de salles et un vocabulaire d'alias plus riche nécessiterait effectivement la capacité d'un

modèle de cette taille. Le cadre schématique actuel constitue une validation de principe dont les mécanismes sont conçus pour passer à l'échelle.

4.6.4 Tableau récapitulatif des performances

La Figure 4.2 présente une synthèse visuelle complète des performances des trois versions à travers l'ensemble des métriques évaluées. Cette vue d'ensemble permet de constater visuellement la progression V1→V2→V2.1 et de comparer simultanément les caractéristiques du corpus, les métriques d'entraînement et les résultats obtenus.

Meilleure performance
Problème résolu
Meilleur ratio

Métrique	V1 Exploratoire	V2 Augmentée	V2.1 Équilibrée
Corpus (train)	1,624	9,206	2,768
Composition	Déséquilibré	Codes explicites	40% aliases
Perte entraînement	0.092	0.186	0.258
Perte validation	0.135	0.315	0.284
Ratio val/train	1.47	1.69	1.10 ✓
PRÉCISION GLOBALE	37.5%	75.0%	93.8% ✓
Identification	0%	100%	100%
Navigation	100%	100%	100%
Aliases	100%*	0%	100% ✓

FIGURE 4.2 – Tableau récapitulatif comparatif des trois versions (V1, V2, V2.1) sur l'ensemble des dimensions évaluées : composition du corpus, métriques d'entraînement, et performances sur l'ensemble de test.

Ce tableau met en évidence plusieurs enseignements clés tirés de la démarche itérative. La précision globale progresse de manière continue à travers les versions, passant de 37,5% (V1) à 75,0% (V2) puis 93,8% (V2.1), démontrant l'efficacité de la stratégie d'amélioration progressive. Le ratio validation/entraînement s'améliore significativement, passant de 1,47 (V1) à 1,10 (V2.1), ce qui témoigne d'une meilleure capacité de généralisation du modèle malgré un corpus plus petit en V2.1 qu'en V2. Les performances par catégorie atteignent toutes 100% en V2.1 pour les trois types principaux (identification directe, navigation classique, *alias* sémantiques), validant l'hypothèse selon laquelle l'équilibrage qualitatif du corpus prime sur l'augmentation quantitative. La durée d'entraînement est optimisée en V2.1 (1h06) par rapport à V2 (3h53), offrant

un gain d'efficacité computationnelle de 72% tout en améliorant les performances de 18,8 points de précision.

4.7 Performances techniques du système

4.7.1 Temps de réponse

Le Tableau 4.9 détaille les temps de réponse du système par phase du pipeline défini à la section 3.6.

TABLE 4.9 – Temps de réponse du système NaviUQTR V2.1 (n=50 requêtes)

Phase	**Médian (ms)**	**Maximum (ms)**
Pattern matching	3,2	12,1
Inférence LLM (identification)	280,0	520,0
Calcul Dijkstra	4,8	8,6
Génération instructions	2,1	5,3
Total	**290,1**	**546,0**

Le temps de réponse médian de 290 millisecondes permet une interactivité fluide. L'inférence LLM constitue l'étape la plus coûteuse (280 ms en médiane), mais demeure acceptable pour une utilisation interactive. Les modules algorithmiques (*pattern matching*, Dijkstra, génération) s'exécutent en moins de 10 ms chacun, confirmant l'efficacité de l'approche hybride présentée à la section 3.2.

4.7.2 Précision des itinéraires

Le calcul d'itinéraires par l'algorithme de Dijkstra garantit mathématiquement l'optimalité du chemin. Sur 50 trajectoires testées, un taux de correspondance exacte de 96% avec les chemins de référence est observé. Les 4% d'écart correspondent à des situations où plusieurs chemins de longueur équivalente existent, l'algorithme sélectionnant un chemin valide mais différent de celui arbitrairement choisi comme référence. Le ratio de longueur moyen atteint 100,8% (écart-type 2,1%), confirmant que même dans ces cas, le surcoût reste négligeable.

Il convient de noter que ce ratio proche de 100% est en partie attribuable à la régularité du graphe schématique utilisé. Dans un graphe en grille régulière, la plupart des chemins entre deux nœuds ont une longueur identique ou très proche, ce qui réduit mécaniquement la variance entre le chemin calculé et le chemin de référence. Dans un bâtiment réel aux topologies irrégulières, cette variance pourrait être plus importante. Cette observation renforce la nécessité d'une validation sur un graphe architectural réel pour confirmer la robustesse de la métrique.

La Figure 3.11 (section 3.6) illustre un exemple concret d'itinéraire optimal calculé par l'algorithme de Dijkstra, confirmant visuellement la cohérence des chemins générés avec la topologie du bâtiment modélisé.

4.7.3 Gestion de l'accessibilité

Le Tableau 4.10 compare les itinéraires générés avec et sans contrainte d'accessibilité selon le mécanisme décrit à la section 3.3.

TABLE 4.10 – Impact de la contrainte d'accessibilité (n=20 trajets multi-étages)

Mode	**Ascenseur**	**Distance (m)**	**Temps (s)**
Standard	65%	42,3	127
Accessible	100%	43,1 (+1,9%)	132 (+3,9%)

L'activation de la préférence d'accessibilité entraîne une utilisation systématique des ascenseurs (100% contre 65% en mode standard), au prix d'un léger allongement du parcours (+1,9%) et du temps (+3,9%). Ce surcoût modéré démontre qu'il est possible de concilier accessibilité et efficacité. Le mécanisme de pondération dynamique des arêtes fonctionne comme prévu, permettant au système d'adapter les itinéraires aux besoins spécifiques de chaque usager.

4.7.4 Empreinte computationnelle et développement durable

Dans un contexte où la consommation énergétique des systèmes d'intelligence artificielle fait l'objet d'une attention croissante, notamment au regard des objectifs de développement durable du Québec et du Canada, il convient d'estimer l'empreinte computationnelle des expériences menées dans ce travail.

Le Tableau 4.11 présente une estimation de la consommation énergétique pour chaque phase d'entraînement, calculée à partir de la puissance nominale de le GPU NVIDIA A100-SXM4-80GB (400W TDP) et des durées d'entraînement mesurées.

TABLE 4.11 – Estimation de la consommation énergétique des phases d'entraînement sur GPU NVIDIA A100-SXM4-80GB (TDP : 400W)

Version	**Durée**	**Puissance**	**Énergie (kWh)**	**CO_2 (g eq.)**
V1	59 min	400 W	0,393	15,7
V2	3h53	400 W	1,553	62,1
V2.1	1h06	400 W	0,440	17,6
Total	**5h58**		**2,387**	**95,5**

Facteur d'émission : 40 g CO_2 eq./kWh (réseau électrique du Québec, hydroélectricité)

Ces estimations révèlent plusieurs constats pertinents. Premièrement, la consommation totale de l'ensemble du processus d'entraînement (2,387 kWh) est comparable à

celle d'un ordinateur portable utilisé pendant une journée de travail, ce qui témoigne de la frugalité computationnelle de l'approche par affinage LoRA par rapport à un entraînement complet. Deuxièmement, la version V2.1, malgré ses meilleures performances, consomme moins d'énergie que V2 (0,440 kWh contre 1,553 kWh), confirmant que l'équilibrage qualitatif du corpus est à la fois plus performant et plus économique. Troisièmement, le facteur d'émission particulièrement faible du réseau électrique québécois (majoritairement hydroélectrique, 40 g CO_2 eq./kWh contre 400-500 g pour les réseaux thermiques) positionne favorablement ce travail au regard des objectifs climatiques du Québec et du Canada. Ces résultats s'inscrivent dans la vision d'une IA responsable et sobre en ressources, alignée avec les principes de la Déclaration de Montréal pour un développement responsable de l'intelligence artificielle.

4.8 Limitations et perspectives

4.8.1 Limitations identifiées

Plusieurs limites circonscrivant actuellement le champ d'application du système ont été identifiées. L'absence de localisation en temps réel constitue la première limitation, le système supposant que l'usager connaît sa position de départ. Le modèle statique du bâtiment nécessite une mise à jour manuelle pour toute modification temporaire (travaux, fermetures). La granularité limitée des repères restreint les instructions aux éléments structurels, sans exploiter les repères visuels spécifiques. La modélisation schématique, bien que suffisante pour valider les principes méthodologiques, diffère d'un bâtiment réel par ses dimensions, sa régularité et le nombre limité de salles représentées. La couverture des *alias* sémantiques constitue une limitation spécifique de l'approche par affinage. Le système ne peut reconnaître que les *alias* explicitement présents dans le corpus d'entraînement. Par exemple, une requête utilisant le terme « bibliothèque » échouerait si cet *alias* n'a pas été inclus dans le corpus V2.1, générant un message d'erreur explicite invitant l'usager à reformuler sa demande. Cette limitation, inhérente aux modèles affinés sur corpus spécialisé, contraste avec les grands modèles pré-entraînés (GPT-4, Claude) qui disposent d'une vaste connaissance sémantique généraliste permettant d'inférer des correspondances non explicitement apprises. Toutefois, elle peut être atténuée par une stratégie d'enrichissement itératif du corpus basée sur les retours utilisateurs. Le processus d'affinage étant rapide (environ 1 heure pour 3,942 exemples), l'ajout régulier de nouveaux *alias* au fil de l'usage constitue une approche viable de maintenance et d'amélioration continue. Cette stratégie transformerait progressivement le système d'un prototype de recherche en un outil opérationnel dont la couverture sémantique s'enrichit organiquement au fil des interactions réelles avec les usagers.

4.8.2 Considérations éthiques et confidentialité

L'intégration de modèles de langage dans des systèmes de navigation intérieure soulève des questions éthiques importantes qui méritent d'être examinées à plusieurs échelles.

TABLE 4.12 – Analyse éthique multi-échelle du système NaviUQTR

Dimension	Enjeu identifié	Mesure adoptée	Perspective
Individuelle	Traçabilité des déplacements via les requêtes utilisateur	Aucune requête stockée dans le prototype actuel	Politique de minimisation des données et consentement explicite
Individuelle	Équité d'accès selon le profil linguistique de l'usager	Corpus incluant des formulations variées	Extension multilingue et évaluation sur différents profils
Individuelle	Accessibilité aux personnes à mobilité réduite	Mécanisme de pondération dynamique favorisant les ascenseurs ($\alpha = 0{,}7$)	Intégration de contraintes d'accessibilité réelles (rampes, portes automatiques)
Institutionnelle	Confidentialité des données architecturales du bâtiment	Modèle schématique sans plans réels ; topologie stockée séparément des poids du LLM	Chiffrement du graphe topologique en déploiement opérationnel
Institutionnelle	Reproductibilité et transparence scientifique	Modèle à source ouverte (Mistral-7B, Apache 2.0) ; corpus synthétique documenté	Publication du code et des corpus sur dépôt public
Institutionnelle	Biais potentiels dans les *alias* sémantiques entraînés	Analyse des erreurs guidant l'enrichissement du corpus	Collecte de retours utilisateurs réels pour diversification
Sociétale	Consommation énergétique de l'entraînement	Affinage LoRA limitant les paramètres entraînables à 2,26% ; réseau québécois majoritairement hydroélectrique	Optimisation des hyperparamètres pour réduire le nombre d'époques nécessaires
Sociétale	Dépendance à des infrastructures numériques centralisées	Architecture hybride fonctionnant sans API externe obligatoire	Déploiement local (*on-premise*) sans dépendance au nuage
Sociétale	Inclusion des populations défavorisées	Interface textuelle accessible sans équipement spécialisé	Extension à l'interface vocale pour les usagers peu familiers avec l'écrit

Cette analyse révèle que NaviUQTR présente plusieurs caractéristiques favorables sur le plan éthique. L'utilisation d'un modèle à source ouverte favorise la transparence et la reproductibilité. L'architecture hybride, qui stocke la topologie du bâtiment séparément des poids du modèle, réduit le risque d'exposition de données architecturales sensibles. La faible empreinte computationnelle de l'affinage LoRA, combinée au contexte énergétique favorable du Québec, limite l'impact environnemental du système.

Confidentialité des données architecturales

Les données utilisées pour entraîner un LLM de navigation intérieure incluent nécessairement des informations sur la topologie du bâtiment (disposition des salles, accès restreints, zones sensibles). Dans le cadre de NaviUQTR, le corpus d'entraînement est entièrement synthétique et basé sur un modèle schématique, ce qui évite toute exposition de données architecturales réelles. Pour un déploiement opérationnel, il conviendrait de s'assurer que les plans détaillés du bâtiment utilisés pour construire le graphe topologique ne soient pas intégrés directement dans les poids du modèle, mais restent dans des systèmes sécurisés séparés. L'architecture hybride de NaviUQTR présente à cet égard un avantage : le LLM n'apprend que des correspondances requête-code, tandis que la topologie complète est stockée dans le graphe NetworkX, un composant distinct pouvant être protégé indépendamment.

Traçabilité des déplacements

Un système de navigation intérieure opérationnel pourrait potentiellement enregistrer les requêtes des usagers, révélant ainsi des informations sur leurs déplacements au sein du bâtiment. Ces données pourraient constituer des informations personnelles au sens de la Loi 25 sur la protection des renseignements personnels dans le secteur privé au Québec. Une implémentation responsable devrait adopter une politique de minimisation des données (ne conserver que ce qui est strictement nécessaire), mettre en place un mécanisme de consentement explicite, et prévoir une anonymisation des requêtes avant tout stockage éventuel.

Biais et équité d'accès

La couverture des *alias* sémantiques dans le corpus d'entraînement peut introduire des biais d'accessibilité : les usagers dont les formulations ne correspondent pas aux patterns entraînés (locuteurs non natifs, personnes utilisant des termes informels ou dialectaux) pourraient obtenir des résultats moins fiables. Une démarche éthique implique de diversifier les formulations du corpus pour couvrir un spectre linguistique

plus large, et d'évaluer explicitement les performances du système sur différents profils d'utilisateurs.

4.8.3 Perspectives à court terme

Plusieurs améliorations peuvent être implémentées rapidement. L'enrichissement des repères visuels dans les instructions améliorerait la confiance de l'usager. L'ajout d'estimations métriques (« environ 30 mètres ») faciliterait l'anticipation du trajet. Un mécanisme de clarification permettrait au système d'engager un micro-dialogue en cas d'ambiguïté. L'extension multilingue, facilitée par la nature multilingue de Mistral, bénéficierait aux étudiants internationaux. La transposition vers un environnement réel nécessiterait l'acquisition de plans officiels, la validation des codes de salles, et l'enrichissement du graphe, sans difficulté conceptuelle majeure.

La gestion des itinéraires multi-destinations constitue également une amélioration prioritaire à court terme. Dans sa version actuelle, NaviUQTR traite chaque requête de manière indépendante, obligeant l'usager à soumettre une nouvelle requête après avoir atteint chaque destination intermédiaire. Un mécanisme de planification séquentielle permettrait de formuler des requêtes du type « Passe d'abord par la salle A-2110, puis rejoins la salle de réunion A-3060 », générant un itinéraire global optimisé en une seule interaction.

L'intégration d'interfaces vocales constitue également une perspective pertinente à court terme. Des API audio-LLM permettant la reconnaissance et la synthèse vocale en français et en anglais sont aujourd'hui disponibles et pourraient être intégrées au pipeline NaviUQTR pour permettre aux visiteurs d'interagir directement dans leur langue sans saisie textuelle. Cette extension serait particulièrement bénéfique dans un contexte universitaire multiculturel, où les étudiants internationaux pourraient formuler leurs requêtes de navigation dans leur langue maternelle. De plus, si une carte intérieure au format BIM (*Building Information Modeling*) est disponible, elle pourrait servir de base pour enrichir le graphe topologique avec des données architecturales réelles, combinant ainsi la flexibilité linguistique du LLM avec une représentation spatiale exhaustive du bâtiment.

4.8.4 Perspectives à moyen terme

Des évolutions plus ambitieuses étendraient significativement les capacités du système. L'intégration de la localisation en temps réel permettrait un guidage dynamique avec recalcul automatique. L'apprentissage à partir de données d'usage réelles affinerait le modèle en identifiant les formulations les plus efficaces. L'extension multi-bâtiments permettrait des itinéraires traversant plusieurs pavillons. L'intégration avec les sys-

tèmes de gestion du bâtiment fournirait des informations contextuelles en temps réel (affluence, état des équipements).

4.9 Conclusion

Ce chapitre a présenté une évaluation complète du système NaviUQTR à travers trois versions successives. La démarche itérative V1→V2→V2.1 a permis d'améliorer progressivement les performances de 37,5% à 93,8%, démontrant l'importance critique de la composition du corpus sur les résultats finaux. Les enseignements méthodologiques majeurs incluent la primauté de l'équilibrage qualitatif sur l'augmentation quantitative, l'importance du ratio validation/entraînement comme indicateur de généralisation, et l'efficacité d'une approche empirique guidée par l'analyse des erreurs. La version finale V2.1 atteint des performances quasi-parfaites sur toutes les catégories de requêtes (100% sur identification, navigation et *alias*), avec un ratio validation/entraînement optimal de 1,10. La comparaison avec les approches existantes positionne favorablement NaviUQTR, offrant une précision supérieure (93,8% vs 85-92%) tout en évitant les dépendances infrastructurelles coûteuses. Le choix de Mistral-7B s'est révélé judicieux, offrant un excellent compromis entre performance, efficacité computationnelle et accessibilité. Les limitations identifiées orientent les développements futurs vers l'enrichissement des repères, l'intégration de la localisation, et la transposition vers des environnements réels complets. Ces perspectives, réalisables à court et moyen terme, transformeraient progressivement NaviUQTR d'un prototype de recherche en un système opérationnel déployable.

Chapitre 5 – Conclusion générale

Dans ce mémoire, nous nous sommes intéressés à la conception d'un système de navigation intérieure capable d'interpréter des requêtes formulées en langage naturel et de générer des itinéraires adaptés à un environnement bâti. Ce travail s'inscrit dans le contexte plus large des bâtiments autonomes, intelligents et communicants, où l'interaction homme–bâtiment constitue un enjeu central.

Le système NaviUQTR que nous avons développé repose sur une approche hybride combinant une modélisation topologique explicite du bâtiment et l'utilisation d'un grand modèle de langage. Notre objectif n'était pas de remplacer les algorithmes classiques de navigation, mais plutôt d'explorer comment un modèle de langage pouvait enrichir l'interprétation des requêtes et améliorer la génération d'instructions compréhensibles pour l'usager.

5.1 Synthèse des travaux réalisés

Les travaux que nous avons réalisés dans ce mémoire s'articulent autour de plusieurs étapes complémentaires. Dans un premier temps, nous avons proposé une représentation du bâtiment sous forme de graphe multi-niveaux, permettant de modéliser les salles, les couloirs et les transitions verticales. Cette modélisation constitue la base de notre système de navigation et garantit la cohérence des itinéraires calculés.

Dans un second temps, nous avons conçu une architecture logicielle modulaire intégrant un agent de navigation chargé du calcul des chemins et un modèle de langage dédié à l'interprétation sémantique des requêtes ainsi qu'à la génération d'instructions textuelles. Cette séparation des rôles s'est révélée essentielle pour assurer la robustesse du système et faciliter son évaluation.

Enfin, nous avons mis en place un processus de spécialisation du modèle Mistral-7B par *affinage*, à l'aide de techniques d'adaptation paramétrique efficaces (LoRA). Nous avons développé trois versions successives du système (V1, V2, V2.1) qui nous ont permis d'observer l'impact de la composition du corpus d'entraînement sur la compréhension des requêtes, la gestion des *alias* sémantiques et la qualité des instructions générées.

5.2 Analyse des résultats

Les résultats obtenus confirment la pertinence de l'approche proposée. La démarche itérative V1→V2→V2.1 a permis d'améliorer la précision de 37,5% à 93,8%, avec un ratio validation/entraînement optimal de 1,10 en V2.1. Les résultats détaillés par version et par catégorie sont présentés au chapitre 4 (section 4.5). Le calcul d'itinéraires par l'algorithme de Dijkstra garantit l'optimalité mathématique des chemins, confirmée par un taux de correspondance exacte de 96% sur 50 trajectoires testées.

L'analyse qualitative des erreurs nous a permis d'identifier les principales sources de difficulté, en particulier les requêtes intrinsèquement ambiguës et les confusions entre codes de salles proches. Ces observations ont contribué à une meilleure compréhension des limites actuelles du système et des mécanismes à renforcer.

5.3 Contributions principales

Ce travail apporte plusieurs contributions au domaine de la navigation intérieure intelligente. Sur le plan méthodologique, nous démontrons empiriquement que l'équilibrage qualitatif du corpus d'entraînement s'avère plus déterminant que l'augmentation quantitative pour le *affinage* de modèles de langage dans des domaines spécialisés. Notre progression V1→V2→V2.1 illustre concrètement ce principe.

Sur le plan technique, nous proposons une architecture hybride fonctionnelle combinant planification algorithmique déterministe (Dijkstra) et compréhension linguistique par LLM *affiné*. Cette approche atteint une précision supérieure (93,8%) aux systèmes purement neuronaux (85-92%) tout en évitant leurs erreurs de raisonnement spatial.

Sur le plan applicatif, nous validons la faisabilité d'un système de navigation intérieure exploitable dans un contexte universitaire réel, capable d'interpréter des requêtes en langage naturel sans infrastructure de localisation coûteuse. Le système NaviUQTR constitue une preuve de concept dont les principes sont transposables à d'autres bâtiments moyennant une phase de modélisation adaptée.

5.4 Limites rencontrées

Malgré les résultats encourageants, notre travail présente certaines limites. Le système suppose que la position initiale de l'usager est connue, ce qui restreint son utilisation dans des contextes réels sans mécanisme de localisation. De plus, le corpus que nous avons utilisé pour l'entraînement demeure de taille modérée (3 942 exemples), ce qui

peut limiter la généralisation du modèle à des formulations très éloignées de celles observées durant l'apprentissage.

Par ailleurs, bien que le bâtiment que nous avons étudié soit inspiré d'un environnement réel, sa modélisation reste schématique. Une représentation plus détaillée, intégrant l'exhaustivité des espaces et les contraintes opérationnelles effectives, serait nécessaire pour un déploiement opérationnel à grande échelle. Toutefois, cette transposition ne présente pas de difficulté conceptuelle majeure, les mécanismes que nous avons développés étant conçus pour s'adapter à tout graphe topologique.

Enfin, notre évaluation repose sur un ensemble de test limité (16 requêtes représentatives). Une validation plus extensive avec des usagers réels dans des conditions d'usage variées permettrait d'identifier des difficultés supplémentaires et d'affiner le système.

5.5 Perspectives

Plusieurs pistes d'évolution peuvent être envisagées à partir de notre travail. À court terme, l'enrichissement du corpus d'entraînement avec des formulations dégradées (erreurs de frappe, abréviations non standard) renforcerait la robustesse face aux requêtes imprécises. L'ajout de repères visuels dans les instructions (« Passez devant le distributeur de café ») améliorerait la confiance de l'usager. L'intégration de mécanismes de désambiguïsation interactifs permettrait au système de clarifier les requêtes ambiguës par un micro-dialogue.

À moyen terme, l'intégration de la localisation en temps réel ouvrirait la voie à un guidage dynamique, capable de s'adapter aux déplacements de l'usager et de recalculer automatiquement l'itinéraire en cas de déviation. L'extension multilingue du système constituerait également une évolution pertinente dans un contexte universitaire international, la nature multilingue de Mistral facilitant cette adaptation.

À plus long terme, la généralisation du système à plusieurs bâtiments interconnectés permettrait de traiter des requêtes du type « Comment aller du pavillon des Sciences au pavillon des Arts ? ». L'intégration avec les systèmes de gestion du bâtiment fournirait des informations contextuelles en temps réel (affluence, état des équipements). La personnalisation avancée des instructions en fonction des préférences individuelles et de l'historique de navigation représente également une perspective intéressante pour les environnements intelligents communicants.

5.6 Réflexion finale

Ce travail nous a permis d'explorer concrètement l'apport des grands modèles de langage dans un système de navigation intérieure. Le développement de NaviUQTR a mis en évidence l'intérêt d'une approche hybride, où les algorithmes classiques garantissent la fiabilité des calculs tandis que les modèles neuronaux apportent souplesse et expressivité linguistique.

Une réflexion s'impose également sur la pertinence d'une approche purement fondée sur un LLM pour la navigation intérieure. En théorie, il serait possible d'énumérer exhaustivement toutes les paires origine-destination possibles et leurs itinéraires correspondants sous forme textuelle, puis d'affiner un LLM pour mémoriser et restituer ces chemins. Cette approche présenterait l'avantage de la simplicité architecturale, en éliminant le module de graphe et l'algorithme de Dijkstra. Cependant, elle souffrirait de plusieurs limitations importantes. Premièrement, elle ne garantit pas l'optimalité des chemins restitués : un LLM peut halluciner des itinéraires plausibles mais sous-optimaux. Deuxièmement, elle ne s'adapterait pas dynamiquement aux modifications du bâtiment (fermetures temporaires, nouvelles contraintes d'accessibilité) sans réentraînement. Troisièmement, elle ne bénéficierait pas des propriétés mathématiques de Dijkstra (complexité $O((V+E)\log V)$, garantie d'optimalité). L'approche hybride retenue dans NaviUQTR constitue donc un choix délibéré fondé sur la complémentarité des paradigmes : expressivité linguistique du LLM pour l'interprétation sémantique, rigueur algorithmique de Dijkstra pour la planification spatiale.

Au-delà du prototype que nous avons développé, ce mémoire contribue à une réflexion plus large sur l'intégration des LLMs dans des environnements physiques structurés. Nous montrons que l'intelligence d'un bâtiment ne repose pas uniquement sur la puissance des modèles utilisés, mais sur la cohérence de l'architecture globale et sur l'articulation entre données, algorithmes et interaction humaine.

La démarche empirique que nous avons adoptée, guidée par l'analyse systématique des erreurs à chaque itération, s'est révélée plus fructueuse qu'une optimisation directe. Cette observation méthodologique dépasse le cadre strict de la navigation intérieure et pourrait inspirer d'autres travaux d'adaptation de LLMs à des domaines spécialisés.

Les compétences que nous avons acquises et les enseignements que nous avons tirés de ce travail constituent une base solide pour des recherches futures dans le domaine des bâtiments intelligents et des systèmes d'assistance basés sur le langage naturel. Nous espérons que ce mémoire encouragera d'autres chercheurs à explorer les nombreuses possibilités qu'offrent les modèles de langage pour améliorer l'interaction entre les usagers et les espaces bâtis.

Références

[1] X. Li, K. Xian, H. Wen, S. Bai, H. Xu et Y. Yu, « PathGen-LLM : A Large Language Model for Dynamic Path Generation in Complex Transportation Networks, » *Mathematics*, t. 13, n° 19, p. 3073, 2025. DOI : 10.3390/math13193073.

[2] J. Chen, B. Lin, R. Xu, Z. Chai, X. Liang et K.-Y. K. Wong, « MapGPT : Map-Guided Prompting with Adaptive Path Planning for Vision-and-Language Navigation, » in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, 2024, p. 1-15.

[3] P. Zhang, H. Liu, Y. Zhu et J. Li, « Enhancing Indoor Navigation with Connected Multi-Graph Networks and Large Language Model Technology, » *Mathematics*, t. 13, n° 19, 2024, DOI à vérifier avant soumission finale.

[4] E. A. Lee, « Cyber Physical Systems : Design Challenges, » in *11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing*, 2008, p. 363-369.

[5] A. Mashhadi et R. Rezvani, « A survey on smart buildings : IoT, AI-based solutions and future trends, » *Journal of Building Engineering*, t. 45, p. 103-118, 2022. DOI : 10.1016/j.jobe.2021.103103.

[6] S. Khan et B. Nizami, « Artificial intelligence-driven energy management in smart buildings, » *Renewable and Sustainable Energy Reviews*, t. 178, p. 113-129, 2023.

[7] L. Xu et J. Wang, « The Internet of Things : State-of-the-art, taxonomy, and future directions, » *ACM Computing Surveys*, t. 56, n° 1, p. 1-42, 2024.

[8] G. Pavlak, « Human-centric AI for intelligent buildings : A review, » in *2025 IEEE International Conference on Smart Infrastructure*, 2025, p. 88-97.

[9] S. Thrun, W. Burgard et D. Fox, *Probabilistic Robotics*. MIT Press, 2005.

[10] P. Boguslawski et L. Mahdjoubi, « Integrated indoor mapping and navigation based on dual graph representation, » *ISPRS Journal of Photogrammetry and Remote Sensing*, t. 116, p. 14-28, 2016.

[11] S. Mohammadi et S. Malek, « An intelligent indoor navigation system using topological maps, » *Expert Systems with Applications*, t. 38, n° 4, p. 3346-3352, 2011.

[12] B. Fleiner et G. Horváth, « Accessible indoor navigation using semantic graphs, » *Journal of Ambient Intelligence and Smart Environments*, t. 8, n° 3, p. 301-320, 2016.

[13] K.-J. Li et J.-S. Lee, *IndoorGML : An Open Standard for Indoor Spatial Information*, Open Geospatial Consortium (OGC) Standard, 2013. adresse : `https://www.ogc.org/standards/indoorgml`.

[14] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid et J. J. Leonard, « Past, present, and future of simultaneous localization and mapping : Towards the robust-perception age, » *IEEE Transactions on Robotics*, t. 32, n° 6, p. 1309-1332, 2016.

[15] X. Wang et J. Gao, « A survey of WiFi-based indoor positioning techniques, » *IEEE Communications Surveys & Tutorials*, t. 18, n° 2, p. 466-490, 2016.

[16] R. Faragher et R. Harle, « Analysis of Bluetooth low energy based indoor positioning accuracy, » in *2014 International Technical Meeting of The Institute of Navigation*, 2014, p. 201-210.

[17] F. Zafari, A. Gkelias et K. K. Leung, « A survey of indoor localization systems and technologies, » *IEEE Communications Surveys & Tutorials*, t. 21, n° 3, p. 2568-2599, 2019.

[18] L. M. Ni, Y. Liu, Y. C. Lau et A. P. Patil, « LANDMARC : Indoor location sensing using active RFID, » in *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, IEEE, 2004, p. 407-415.

[19] E. W. Dijkstra, « A note on two problems in connexion with graphs, » *Numerische Mathematik*, t. 1, n° 1, p. 269-271, 1959.

[20] P. E. Hart, N. J. Nilsson et B. Raphael, « A formal basis for the heuristic determination of minimum cost paths, » *IEEE Transactions on Systems Science and Cybernetics*, t. 4, n° 2, p. 100-107, 1968.

[21] T. Pfeifer et G. Retscher, « Multi-criteria indoor pathfinding for accessibility, » *Sensors*, t. 20, n° 4, p. 1012, 2020.

[22] S. Jung et J. Lee, « Indoor navigation with smartphone sensors, » *IEEE Sensors Journal*, t. 15, n° 4, p. 2333-2348, 2015.

[23] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould et A. van den Hengel, « Vision-and-Language Navigation : Interpreting visually-grounded navigation instructions in real environments, » in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, p. 3674-3683.

[24] Y. Li, J. Wang, X. Liang, J. Jia et J. Shao, « VLN-BERT : A Recurrent Vision-and-Language BERT for Navigation, » *IEEE Transactions on Pattern Analysis and Machine Intelligence*, t. 45, nº 6, p. 7695-7710, 2022.

[25] S. Chen, P.-L. Guhur, C. Schmid et I. Laptev, « HAMT : History Aware Multimodal Transformer for Vision-and-Language Navigation, » in *Advances in Neural Information Processing Systems*, t. 34, 2021, p. 5834-5846.

[26] J. Krantz, A. Gokaslan, D. Batra, S. Lee et O. Maksymets, « Sim-to-Real Transfer for Vision-and-Language Navigation, » *arXiv preprint arXiv :2204.11109*, 2022.

[27] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver et K. Kavukcuoglu, « Asynchronous Methods for Deep Reinforcement Learning, » t. 48, p. 1928-1937, 2016.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford et O. Klimov, « Proximal Policy Optimization Algorithms, » *arXiv preprint arXiv :1707.06347*, 2017.

[29] F. Hill, A. Lampinen, R. Schneider, S. Clark, M. Botvinick, J. L. McClelland et A. Santoro, « Environment Generation for Zero-Shot Compositional Learning, » *Advances in Neural Information Processing Systems*, t. 33, 2020.

[30] M. Chevalier-Boisvert, D. Bahdanau, S. Lahlou, L. Willems, C. Saharia, T. H. Nguyen et Y. Bengio, « BabyAI : A Platform to Study the Sample Efficiency of Grounded Language Learning, » in *International Conference on Learning Representations*, 2019.

[31] M. Shridhar, J. Thomason, D. Gordon, Y. Bisk, W. Han, R. Mottaghi, L. Zettlemoyer et D. Fox, « ALFRED : A Benchmark for Interpreting Grounded Instructions for Everyday Tasks, » in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020, p. 10 740-10 749.

[32] C. Li, R. Zhang, J. Wong, C. Gokmen, S. Srivastava, R. Martín-Martín, C. Wang, G. Levine, M. Lingelbach, J. Sun et al., « BEHAVIOR-1K : A Benchmark for Embodied AI with 1,000 Everyday Activities and Realistic Simulation, » in *Conference on Robot Learning*, 2024.

[33] A. Kalyuzhnaya, S. Mityagin, E. Lutsenko, A. Getmanov, Y. Aksenkin, K. Fatkhiev, K. Fedorin, N. O. Nikitin, N. Chichkova, V. Vorona et A. Boukhanovsky, « LLM Agents for Smart City Management : Enhancing Decision Support Through Multi-Agent AI Systems, » *Smart Cities*, t. 8, nº 1, p. 19, 2025. doi : 10.3390/smartcities8010019.

[34] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell et al., « Language Models are Few-Shot Learners, » *Advances in Neural Information Processing Systems*, t. 33, p. 1877-1901, 2020.

[35] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts et al., « PaLM : Scaling Language Modeling with Pathways, » *Journal of Machine Learning Research*, t. 24, p. 1-113, 2023.

[36] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar et al., « LLaMA : Open and Efficient Foundation Language Models, » *arXiv preprint arXiv :2302.13971*, 2023.

[37] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier et al., « Mistral 7B, » *arXiv preprint arXiv :2310.06825*, 2023.

[38] I. Papaioannou, C. Korkas et E. Kosmatopoulos, « Smart Building Recommendations with LLMs : A Semantic Comparison Approach, » *Buildings*, t. 15, n° 13, p. 2303, 2025. doi : 10.3390/buildings15132303.

[39] D. Patel et A. Balasubramanian, « Evaluating the Spatial Reasoning Abilities of Large Language Models, » *arXiv preprint arXiv :2306.00988*, 2023.

[40] V. Dorbala, J. Biswas et D. Garg, « Wayfinding with Large Language Models, » *arXiv preprint arXiv :2401.05271*, 2024.

[41] Y. Mao, J. Zhong, C. Fang, J. Zheng, R. Tang, H. Zhu, P. Tan et Z. Zhou, « SPATIALLM : Training Large Language Models for Structured Indoor Modeling, » *arXiv preprint arXiv :2506.07491*, 2025, Version 2, cs.CV.

[42] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang et W. Chen, « LoRA : Low-Rank Adaptation of Large Language Models, » in *International Conference on Learning Representations*, 2022. adresse : https://openreview.net/forum?id=nZeVKeeFYf9.

[43] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu et al., « PaLM-E : An Embodied Multimodal Language Model, » *arXiv preprint arXiv :2303.03378*, 2023.