REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
الجمهورية الجزائرية الديمقراطية الشعبية

MINISTRY OF HIGHER EDUCATION
AND SCIENTIFIC RESEARCH

HIGHER SCHOOL IN APPLIED SCIENCES
–T L E M C E N–



المدرسة العليا في العلوم التطبيقية
École Supérieure en
Sciences Appliquées

وزارة التعليم العالي والبحث العلمي

المدرسة العليا في العلوم التطبيقية
- تلمسان -

Mémoire de fin d'étude

Pour l'obtention du diplôme d'Ingénieur

Filière : Génie industriel
Spécialité : Management industriel et logistique

Présenté par : Rym TALEB

Thème

Évaluation du potentiel des grands modèles de langage (LLM) pour résoudre des problèmes d'optimisation combinatoire

Soutenu publiquement, le 18 / 06 / 2026 , devant le jury composé de :

Mohammed Benekfrouf	Maître de conférences A	ESSA Tlemcen	Président
Fouad Maliki	Maître de conférences A	ESSA Tlemcen	Directeur de mémoire
Nadjib Brahimi	Professeur	IMT Atlantique / Rennes SB	Co-directeur de mémoire
Mehdi Souier	Professeur	Université de Tlemcen	Examinateur 1
Abdelghani Bekrar	Maître de conférences	UPHF	Examinateur 2
Sadeque Hamdan	Maître de conférences	Bangor University / Bangor BS	Invité

Année universitaire :2025/2026


بسم الله الرحمن الرحيم

Dédicaces

Je dédie ce travail à mes chers parents, pour leur amour, leur soutien inconditionnel, leur patience et leurs encouragements constants tout au long de mon parcours. Depuis mon enfance jusqu'à l'aboutissement de cette étape importante de ma formation, leur présence, leurs sacrifices et leur confiance ont été une source essentielle de force et de motivation.

Je dédie également ce mémoire à mes sœurs et à mes beaux-frères, pour leur soutien, leur bienveillance et leurs encouragements. Leur présence à mes côtés a toujours représenté pour moi un appui précieux dans les moments de doute comme dans les moments de réussite.

À mes petits neveux Ahmed et Mohamed Mounir, ainsi qu'à ma petite nièce Asma, dont l'innocence, la joie et l'affection apportent beaucoup de bonheur à ma vie. Leur présence me rappelle l'importance de persévérer et de construire un avenir dont ma famille pourra être fière.

Je souhaite également dédier ce mémoire à Monsieur Fouad Maliki, mon encadrant académique et responsable de formation, qui a profondément marqué mon parcours. Après des années de classes préparatoires particulièrement difficiles, il a su me faire découvrir et aimer le Génie Industriel. Par sa pédagogie, son accompagnement, sa bienveillance et son exigence, il a été pour moi bien plus qu'un enseignant : l'un des meilleurs professeurs que j'ai eu la chance de rencontrer tout au long de mon parcours.

Je souhaite également adresser une pensée particulière à Monsieur Nadjib Brahimi, mon encadrant de stage, qui m'a fait confiance et m'a donné l'opportunité de rejoindre cette expérience de recherche. Cette confiance a représenté pour moi une véritable source de motivation et m'a permis de me sentir pleinement impliquée dans ce travail. Son accompagnement, sa disponibilité et ses encouragements ont contribué à rendre cette expérience à la fois enrichissante, formatrice et marquante dans mon parcours.

J'adresse également une pensée sincère à mes amies Kawther, Hind, Sara et Ghizlene, avec lesquelles j'ai partagé cette formation en Génie Industriel. Je les remercie pour leur amitié, leur soutien, les moments partagés et les souvenirs précieux construits tout au long de ce parcours.

Enfin, je dédie ce mémoire à toutes les personnes qui ont cru en moi, qui m'ont encouragée et qui m'ont accompagnée, de près ou de loin, jusqu'à l'obtention de ce diplôme.

Rym

Remerciements

Au terme de ce travail, je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué, de près ou de loin, à la réalisation de ce mémoire.

Je souhaite tout d'abord remercier les membres du jury pour l'honneur qu'ils me font en acceptant d'évaluer ce travail. Je leur suis reconnaissante pour le temps consacré à la lecture de ce mémoire, ainsi que pour leurs remarques et recommandations qui contribueront à enrichir ma réflexion.

J'adresse mes sincères remerciements à mon encadrant académique, Monsieur Fouad Maliki, pour son suivi, ses remarques constructives, son soutien moral et ses encouragements durant la réalisation de ce mémoire. Ses conseils ont contribué à améliorer la cohérence scientifique et méthodologique de ce travail, et son accompagnement m'a été précieux tout au long de cette période.

Je remercie également Monsieur Nadjib Brahimi, mon encadrant de stage, pour son accompagnement, sa disponibilité, ses conseils et la confiance qu'il m'a accordée tout au long de ce travail. Je le remercie particulièrement de m'avoir donné l'opportunité de rejoindre cette expérience de recherche et de m'avoir guidée avec bienveillance et exigence.

J'adresse également ma profonde reconnaissance et mes sincères remerciements à Monsieur Sadeque Hamdan pour son aide, ses échanges et son appui tout au long de ce travail. Son accompagnement, ses remarques et sa disponibilité ont été précieux pour renforcer la qualité de ma démarche et consolider les résultats présentés dans ce mémoire.

Je tiens également à remercier l'IMT Atlantique, le laboratoire LS2N ainsi que l'équipe Modelis pour leur accueil et pour l'environnement de travail stimulant dont j'ai pu bénéficier durant mon stage. Cette expérience m'a permis de découvrir davantage le monde de la recherche.

Mes remerciements s'adressent aussi à l'École Supérieure en Sciences Appliquées de Tlemcen, ainsi qu'à l'ensemble des enseignants qui ont contribué à ma formation en Génie Industriel. Les connaissances acquises tout au long de mon parcours ont constitué une base essentielle pour mener à bien ce travail.

Enfin, j'adresse mes remerciements les plus profonds à ma famille pour son soutien constant, sa patience, ses encouragements et sa confiance. Je remercie également mes amis et toutes les personnes qui m'ont accompagnée, soutenue ou encouragée durant cette période.

À toutes ces personnes, j'adresse mes plus sincères remerciements.

ملخص

تركز هذه المذكرة على تقييم النماذج اللغوية الكبيرة في سياق التحسين التوافقي المطبق على اللوجستيك وإدارة الإنتاج. الهدف الرئيسي من هذا العمل هو تحليل مدى قدرة هذه النماذج على المساعدة في حل مسائل التحسين، خاصة عندما يتم استخدامها من طرف مستخدم غير متخصص.

تعتمد الدراسة على ثلاث مسائل كلاسيكية في التحسين التوافقي، وهي مسألة الوسيط غير المقيدة، ومسألة تغطية المجموعات، ومسألة تحديد أحجام الدفعات غير المقيدة. تم اختيار هذه المسائل نظرا لأهميتها في بحوث العمليات، وارتباطها المباشر بقرارات صناعية ولوجستية، إضافة إلى توفر أمثلة معيارية وحلول مرجعية تسمح بتقييم النتائج. وقد شملت الدراسة خمسة نماذج لغوية كبيرة، وهي: شات جي بي تي، جيميني، ديب سيك، غروك، وكلود.

تبدأ المذكرة بعرض المفاهيم الأساسية المتعلقة بالذكاء الاصطناعي، والذكاء الاصطناعي التوليدي، والنماذج اللغوية الكبيرة، ثم تقدم مبادئ هندسة الأوامر. بعد ذلك، يتم اقتراح إطار منهجي لتصميم الأوامر الموجهة إلى النماذج، يعتمد على عدة مبادئ، من بينها استعمال اللغة الإنجليزية، تحديد دور خبير للنموذج، منع إعادة استعمال الحلول المثلى المعروفة، تقسيم المهمة إلى خطوات واضحة، تحديد شكل المخرجات المطلوبة، وقياس زمن الاستجابة.

تهدف المرحلة التجريبية الأولى إلى تقييم النماذج اللغوية الكبيرة كأدوات حل مستقلة. في هذه المرحلة، تحصل النماذج على وصف المسألة والمعطيات المرتبطة بها دون تزويدها بطريقة حل مفصلة أو ببرنامج تحقق خارجي. بعد ذلك، يتم فحص الإجابات باستعمال برامج مكتوبة بلغة بايثون، ثم تنظيم النتائج وتحليلها باستخدام مايكروسوفت إكسل. وقد تم الاعتماد على عدة مؤشرات للتقييم، من بينها جودة الحل، المثالية في أول محاولة، القدرة، قابلية التنفيذ، الموثوقية، الاستقرار، القدرة على التحسن، كفاءة عدد الطلبات، زمن الاستجابة، والفجوة مقارنة بالحل المرجعي.

أظهرت النتائج أن النماذج اللغوية الكبيرة قادرة على إنتاج إجابات منظمة، وأحيانا حلول ذات جودة جيدة. غير أن أداءها يختلف بشكل واضح حسب النموذج، ونوع المسألة، وحجم المثال المدروس. تميز نموذج كلود عموما من حيث جودة الحل، وقابلية التنفيذ، والموثوقية، خاصة في مسألة الوسيط ومسألة تحديد أحجام الدفعات. كما أظهر نموذج غروك نتائج جيدة في مسألة تغطية المجموعات، إضافة إلى قدرة على التحسن في بعض الحالات. أما شات جي بي تي، فكان غالبا الأسرع في زمن الاستجابة، لكن حلوله تحتاج إلى تحقق دقيق. في المقابل، أظهرت نماذج جيميني وديب سيك أداء أقل انتظاما، خاصة من حيث قابلية التنفيذ والموثوقية.

تبرز هذه الدراسة محدودية مهمة عند استخدام النماذج اللغوية الكبيرة كأدوات حل مستقلة، إذ إن الإجابة المصاغة بشكل جيد لا تعني بالضرورة أن الحل صحيح. فقد ترتكب هذه النماذج أخطاء في الحساب، أو تقترح حلولا غير قابلة للتنفيذ، أو تعلن خطأ أن الحل أمثل. لذلك، لا يمكن اعتبار هذه النماذج بديلا عن طرق التحسين الكلاسيكية أو عن برامج الحل المتخصصة. ومع ذلك، يمكن أن تكون أدوات مساعدة مفيدة في صياغة المسائل، واقتراح حلول أولية، وشرح منهجيات الحل، ومرافقة المستخدم في عملية التحليل، بشرط أن يتم التحقق من نتائجها بشكل منهجي.

الكلمات المفتاحية: الذكاء الاصطناعي التوليدي، النماذج اللغوية الكبيرة، التحسين التوافقي، بحوث العمليات، هندسة الأوامر، مسألة الوسيط، تغطية المجموعات، تحديد أحجام الدفعات، قابلية التنفيذ، الموثوقية.

Abstract

This thesis focuses on the evaluation of Large Language Models (LLMs) in the context of combinatorial optimization applied to logistics and production management. The main objective is to analyze to what extent general-purpose LLMs can be used as support tools for solving optimization problems, particularly when they are used by non-expert users.

The study considers three classical optimization problems: the *Uncapacitated P-Median Problem* (UPMP), the *Set Covering Problem* (SCP), and the *Uncapacitated Lot Sizing Problem* (ULS). These problems were selected because of their relevance in operations research, their connection to industrial and logistics decision-making, and the availability of benchmark instances and reference optimal solutions. Five LLMs were evaluated: ChatGPT 5.3, Gemini 3, DeepSeek, Grok 4, and Claude Sonnet 4.6.

The thesis first introduces the main concepts related to artificial intelligence, generative AI, and LLMs, then presents the principles of prompt engineering. A prompt engineering framework is then proposed to structure the requests submitted to the models. This framework is based on several principles, including the use of English prompts, explicit expert role specification, the prohibition of directly reproducing known optimal solutions, task decomposition, a clearly defined output format, and response time measurement.

The first experimental phase evaluates LLMs as autonomous solvers. The models receive the problem description and the associated data, without being provided with an external solver or an imposed solution method. The generated answers are then verified using Python scripts and analyzed with Microsoft Excel. Several indicators are used, including *Solution Quality*, *Optimality First Iteration*, *Capability*, *Feasibility*, *Reliability*, *Stability*, *Improve Learning*, *Query Efficiency*, *Response Time*, and *GAP*.

The results show that LLMs are able to produce structured answers and, in some cases, good-quality solutions. However, their performance varies significantly depending on the model, the problem, and the size of the instance. Claude Sonnet 4.6 generally stands out in terms of solution quality, feasibility, and reliability, particularly for the p-median problem and ULS. Grok 4 obtains strong results on the SCP and shows a good improvement capability in some cases. ChatGPT 5.3 is often the fastest model, but its solutions require careful verification. Gemini 3 and DeepSeek show more irregular performance, especially regarding feasibility and reliability.

This study highlights an important limitation of LLMs used as autonomous solvers: a well-written answer does not guarantee a correct solution. The models may make calculation errors, generate infeasible solutions, or incorrectly claim that a solution is optimal. Therefore, LLMs cannot replace classical optimization methods or specialized solvers. However, they can be useful assistants for formulating problems, proposing initial solutions, explaining solution approaches, or supporting users during the resolution process, provided that their outputs are systematically verified.

Keywords: generative artificial intelligence, Large Language Models, combinatorial optimization, operations research, prompt engineering, p-median, set covering, lot sizing, feasibility, reliability.

Résumé

Ce mémoire s'intéresse à l'évaluation des grands modèles de langage, ou *Large Language Models* (LLM), dans le contexte de l'optimisation combinatoire appliquée à la logistique et à la gestion de production. L'objectif principal est d'analyser dans quelle mesure des LLM grand public peuvent être utilisés comme outils d'aide à la résolution de problèmes d'optimisation, en particulier lorsqu'ils sont sollicités par un utilisateur non expert.

L'étude porte sur trois problèmes classiques : le *Uncapacitated P-Median Problem* (UPMP), le *Set Covering Problem* (SCP) et le *Uncapacitated Lot Sizing Problem* (ULS). Ces problèmes ont été choisis pour leur importance en recherche opérationnelle, leur lien direct avec des décisions industrielles et logistiques, ainsi que pour la disponibilité d'instances de benchmark et de solutions de référence. Cinq LLM ont été évalués : ChatGPT 5.3, Gemini 3, DeepSeek, Grok 4 et Claude Sonnet 4.6.

Le travail commence par une présentation des notions liées à l'intelligence artificielle, à l'IA générative et aux LLM, puis introduit les principes du *prompt engineering*. Un framework de conception de prompts est ensuite proposé afin de structurer les demandes soumises aux modèles. Ce framework repose notamment sur la formulation en anglais, la spécification d'un rôle d'expert, l'interdiction de reproduire des solutions optimales connues, la décomposition des tâches, la demande d'un format de sortie précis et la mesure du temps de réponse.

La première phase expérimentale évalue les LLM comme solveurs autonomes. Les modèles reçoivent la description du problème et les données associées, sans solveur externe ni méthode de résolution imposée. Les réponses produites sont ensuite vérifiées à l'aide de scripts Python et analysées dans Excel. Plusieurs indicateurs sont utilisés, notamment *Solution Quality*, *Optimality First Iteration*, *Capability*, *Feasibility*, *Reliability*, *Stability*, *Improve Learning*, *Query Efficiency*, *Response Time* et *GAP*.

Les résultats montrent que les LLM peuvent produire des réponses structurées et parfois des solutions de bonne qualité, mais que leurs performances varient fortement selon le modèle, le problème et la taille des instances. Claude Sonnet 4.6 se distingue globalement par sa qualité de solution, sa faisabilité et sa fiabilité, notamment sur le p-median et le ULS. Grok 4 obtient de très bons résultats sur le SCP et présente une bonne capacité d'amélioration dans certains cas. ChatGPT 5.3 est souvent le plus rapide, mais ses solutions nécessitent une vérification attentive. Gemini 3 et DeepSeek montrent des performances plus irrégulières, en particulier en matière de faisabilité et de fiabilité.

Cette étude met en évidence une limite importante des LLM utilisés comme solveurs autonomes : une réponse bien formulée ne garantit pas une solution correcte. Les modèles peuvent commettre des erreurs de calcul, produire des solutions infaisables ou annoncer à tort une solution optimale. Ainsi, les LLM ne peuvent pas remplacer les méthodes classiques d'optimisation ni les solveurs spécialisés. En revanche, ils peuvent constituer des assistants utiles pour formuler un problème, proposer une première solution, expliquer une démarche ou accompagner un utilisateur dans un processus de résolution, à condition que leurs résultats soient systématiquement vérifiés.

Mots-clés : intelligence artificielle générative, grands modèles de langage, optimisation combinatoire, recherche opérationnelle, prompt engineering, p-median, set covering, lot sizing, faisabilité, fiabilité.

Table des matières

Dédicaces **2**

Remerciements **3**

3

4

Abstract **5**

Abstract **5**

Résumé **6**

Résumé **6**

Liste des figures **12**

Liste des tableaux **14**

Liste des abréviations **15**

Introduction générale **17**

Introduction générale **18**

1 Intelligence artificielle : concepts, couches technologiques et émergence des LLM **20**

1.1 Introduction générale 20

1.2 Clarification des couches : IA, ML, DL, IA générative et LLM 21

1.2.1 L'IA : la notion « parapluie » 22

1.2.2 Machine learning : apprendre à partir de données 22

1.2.3 Deep learning : réseaux neuronaux profonds 23

1.2.4 IA générative : produire du contenu nouveau 23

1.2.5 LLM : modèles de langage à grande échelle 23

1.2.6 Synthèse hiérarchique 24

1.3 Approches de l'IA : symbolique, statistique et hybridation 24

1.3.1 IA symbolique 24

1.3.2 IA statistique ou connexionniste . . . 25
1.3.3 Hybridation : une approche naturelle pour des tâches à contraintes 25
1.4 Machine learning : fondations utiles pour l'optimi- sation . . . 25
1.4.1 Apprentissage supervisé . . . 25
1.4.2 Apprentissage non supervisé . . . 26
1.4.3 Apprentissage par renforcement . . . 26
1.4.4 Limites du ML en présence de contraintes . . . 27
1.5 Deep learning : pourquoi il a ouvert la voie aux LLM . . . 27
1.5.1 Le neurone artificiel et le perceptron . . . 27
1.5.2 Architectures clés et leur pertinence . . . 27
1.5.3 Le mécanisme d'attention : fondateur des LLM . . . 28
1.5.4 Architecture Transformer et encodage positionnel . . . 28
1.5.5 Coût, robustesse et interprétabilité . . . 28
1.6 Grands modèles de langage (LLM) : fonctionnement, capacités et limites 28
1.6.1 Définition et notion de tokens . . . 28
1.6.2 Étapes d'entraînement des LLM . . . 29
1.6.3 Capacités émergentes des LLM . . . 29
1.6.4 Capacités utiles en optimisation combinatoire . . . 30
1.6.5 Paramètres de génération et stabilité . . . 30
1.6.6 Limites critiques des LLM en présence de contraintes . . . 30
1.7 Conclusion . . . 31

2 *Prompt Engineering* : techniques, cadres et applications 32
2.1 Introduction . . . 32
2.2 Fondements conceptuels du *Prompt Engineering* . . . 33
2.2.1 Définition et positionnement . . . 33
2.2.2 Sensibilité des LLM aux prompts . . . 33
2.2.3 Évolution du domaine . . . 33
2.3 Anatomie d'un prompt : composantes et structure . . . 34
2.3.1 Composantes fondamentales . . . 34
2.3.2 Structure et ordonnancement des composantes . . . 34
2.3.3 Paramètres de génération . . . 35
2.4 Taxonomie des techniques de *Prompt Engineering* . . . 35
2.4.1 Techniques de base . . . 35
2.4.2 Techniques de raisonnement structuré . . . 36
2.4.3 Techniques de structuration et de contrôle . . . 37
2.4.4 Techniques avancées . . . 37
2.5 Catalogues de patterns et cadres méthodologiques . . . 38
2.5.1 Le catalogue de patterns de White et al. . . . 38
2.5.2 Le framework Impromptu de Morales et al. . . . 39
2.6 *Prompt Engineering* pour l'optimisation mathématique . . . 39
2.6.1 Erreurs de modélisation fréquentes . . . 40
2.6.2 Stratégies de prompting efficaces . . . 40
2.6.3 Évaluation des réponses en optimisation . . . 41
2.7 Métriques d'évaluation des prompts . . . 41
2.7.1 Métriques basées sur la sortie . . . 41
2.7.2 Métriques intrinsèques au prompt . . . 41
2.7.3 *Prompt Efficiency* . . . 42

2.8 Limites du *Prompt Engineering* et défis ouverts . . . 42
2.8.1 Fragilité des prompts . . . 42
2.8.2 Manque de robustesse . . . 42
2.8.3 Absence de garantie formelle . . . 42
2.8.4 Scalabilité des instances . . . 42
2.8.5 Nécessité d'une vérification externe . . . 43
2.9 Conclusion . . . 43

3 Framework de *Prompt Engineering* pour l'optimisation combinatoire 44
3.1 Introduction . . . 44
3.2 Observations préliminaires ayant guidé la conception . . . 44
3.2.1 Observation 1 : la langue du prompt influence significativement la qualité des réponses . . . 45
3.2.2 Observation 2 : la spécification explicite d'un rôle d'expert améliore la rigueur mathématique . . . 45
3.2.3 Observation 3 : l'interdiction des solutions optimales connues est indispensable pour un raisonnement authentique . . . 46
3.3 Les sept principes fondateurs du framework . . . 46
3.3.1 Vue d'ensemble . . . 46
3.3.2 Principe 1 : rôle d'expert explicite . . . 47
3.3.3 Principe 2 : benchmark standardisé et format d'instance explicite 47
3.3.4 Principe 3 : interdiction des solutions mémorisées . . . 48
3.3.5 Principe 4 : décomposition structurée de la tâche . . . 48
3.3.6 Principe 5 : spécification explicite du format de sortie . . . 48
3.3.7 Principe 6 : mesure explicite du temps de réponse . . . 48
3.3.8 Principe 7 : combinaison de styles de prompting . . . 49
3.4 Conception et analyse des prompts . . . 49
3.4.1 Architecture commune aux trois prompts . . . 49
3.4.2 Prompt pour le problème du p-médian . . . 50
3.4.3 Prompt pour le *Set Covering Problem* . . . 51
3.4.4 Prompt pour le problème *Uncapacitated Lot Sizing* . . . 52
3.5 Comparaison des trois prompts et cohérence du framework . . . 54
3.5.1 Tableau comparatif des composantes . . . 54
3.5.2 Cohérence et invariants du framework . . . 54
3.5.3 Positionnement du framework proposé par rapport aux travaux existants . . . 54
3.6 Conclusion . . . 55

4 Évaluation des LLM comme solveurs autonomes 56
4.1 Introduction . . . 56
4.2 Problématique et objectifs . . . 57
4.2.1 Contexte et motivation . . . 57
4.2.2 Question de recherche . . . 57
4.2.3 Objectifs spécifiques . . . 57
4.3 Protocole expérimental . . . 58
4.3.1 Vue d'ensemble du protocole . . . 58
4.3.2 Design expérimental . . . 58
4.4 Benchmarks et problèmes étudiés . . . 59
4.4.1 Uncapacitated P-Median Problem (UPMP) . . . 59

4.4.2 Set Covering Problem (SCP) . . . 60
4.4.3 Uncapacitated Lot Sizing Problem (ULS) . . . 61
4.5 LLM évalués . . . 62
4.6 Indicateurs d'évaluation . . . 63
4.6.1 Qualité de la solution . . . 64
4.6.2 Faisabilité et validité . . . 64
4.6.3 Raisonnement et comportement . . . 65
4.6.4 Efficacité pratique . . . 65
4.7 Outils utilisés . . . 65
4.8 Résultats expérimentaux du problème p-median . . . 65
4.8.1 Comparaison des GAP moyens . . . 66
4.8.2 Analyse de l'optimalité . . . 67
4.8.3 Analyse de la fiabilité . . . 68
4.8.4 Amélioration au fil des itérations . . . 68
4.8.5 Temps de réponse . . . 69
4.8.6 Comparaison globale des LLM sur le p-median . . . 69
4.8.7 Effet de la taille des instances . . . 71
4.9 Résultats expérimentaux du problème *Set Covering Problem* . . . 71
4.9.1 Comparaison des GAP moyens . . . 72
4.9.2 Analyse de l'optimalité . . . 72
4.9.3 Analyse de la faisabilité . . . 73
4.9.4 Analyse de la fiabilité . . . 74
4.9.5 Amélioration au fil des itérations . . . 74
4.9.6 Temps de réponse . . . 75
4.9.7 Analyse de la stabilité . . . 76
4.9.8 Efficacité en nombre de requêtes . . . 76
4.9.9 Comparaison globale des LLM sur le SCP . . . 77
4.9.10 Effet de la taille des instances . . . 78
4.10 Résultats expérimentaux du problème ULS . . . 78
4.10.1 Comparaison des GAP moyens . . . 79
4.10.2 Analyse de l'optimalité . . . 80
4.10.3 Analyse de la faisabilité . . . 80
4.10.4 Analyse de la fiabilité . . . 81
4.10.5 Amélioration au fil des itérations . . . 82
4.10.6 Temps de réponse . . . 82
4.10.7 Analyse de la stabilité . . . 83
4.10.8 Efficacité en nombre de requêtes . . . 84
4.10.9 Comparaison globale des LLM sur le ULS . . . 84
4.10.10 Effet de la taille des instances . . . 85
4.11 Conclusion . . . 86

Conclusion générale **88**

Conclusion générale **88**

Annexes **90**

A Prompts utilisés dans les expérimentations **90**
A.1 Prompt du problème p-median 90
A.1.1 Prompt avec spécification de rôle 90
A.1.2 Prompt sans spécification de rôle 92
A.2 Prompt du *Set Covering Problem* 93
A.2.1 Prompt avec spécification de rôle 93
A.2.2 Prompt sans spécification de rôle 95
A.3 Prompt du problème *Uncapacitated Lot Sizing* 96
A.3.1 Prompt avec spécification de rôle 96
A.3.2 Prompt sans spécification de rôle 98

B Instances de benchmark utilisées **101**
Sources des benchmarks 101
B.1 Instances du problème p-median 101
B.2 Instances du *Set Covering Problem* 102
B.3 Instances du problème *Uncapacitated Lot Sizing* 103

Bibliographie **105**

Table des figures

1.1 Hiérarchie des notions : IA, ML, DL, IA générative et LLM. 22

3.1 Architecture du framework de *prompt engineering* : des observations aux prompts. 47

Liste des tableaux

1.1 Hiérarchie des notions IA et pertinence pour l'optimisation combinatoire 24
1.2 Capacités des LLM applicables à l'optimisation combinatoire [29] . . . 30

2.1 Exemples de patterns de prompts utiles pour l'optimisation combinatoire 39

3.1 Comparaison des composantes des trois prompts conçus 54

4.1 Design expérimental de la Phase 1 58
4.2 Instances UPMP utilisées 60
4.3 Instances SCP utilisées 61
4.4 Instances ULS utilisées 62
4.5 LLM évalués dans la Phase 1 63
4.6 Indicateurs liés à la qualité de la solution 64
4.7 Indicateurs liés à la faisabilité et à la validité 64
4.8 Indicateurs liés au raisonnement et au comportement 65
4.9 Indicateurs liés à l'efficacité pratique 65
4.10 Comparaison des GAP moyens pénalisés des LLM sur le problème du p-median 66
4.11 Nombre de solutions optimales obtenues par les LLM sur le problème du p-median 67
4.12 Comparaison de la fiabilité des LLM sur le problème du p-median . . . 68
4.13 Amélioration des LLM au fil des itérations sur le problème du p-median 68
4.14 Classement des LLM selon le temps de réponse sur le problème du p-median 69
4.15 Comparaison globale des LLM sur le problème du p-median 70
4.16 Comparaison des GAP moyens pénalisés des LLM sur le problème SCP 72
4.17 Nombre de solutions optimales obtenues par les LLM sur le problème SCP 73
4.18 Comparaison de la faisabilité des solutions proposées par les LLM sur le SCP 73
4.19 Comparaison de la fiabilité des LLM sur le problème SCP 74
4.20 Amélioration des LLM au fil des itérations sur le problème SCP 75
4.21 Classement des LLM selon le temps de réponse sur le problème SCP . . 75
4.22 Comparaison de la stabilité des LLM sur le problème SCP 76
4.23 Efficacité des LLM en nombre de requêtes sur le problème SCP 76
4.24 Comparaison globale des LLM selon l'ensemble des indicateurs pour le SCP 77

4.25 Comparaison des GAP moyens pénalisés des LLM sur le problème ULS 79
4.26 Nombre de solutions optimales obtenues par les LLM sur le problème ULS 80
4.27 Comparaison de la faisabilité des solutions proposées par les LLM sur le problème ULS 81
4.28 Comparaison de la fiabilité des LLM sur le problème ULS 81
4.29 Amélioration des LLM au fil des itérations sur le problème ULS 82
4.30 Classement des LLM selon le temps de réponse sur le problème ULS . . 83
4.31 Comparaison de la stabilité des LLM sur le problème ULS 83
4.32 Efficacité des LLM en nombre de requêtes sur le problème ULS 84
4.33 Comparaison globale des LLM selon l'ensemble des indicateurs pour le problème ULS 84
4.34 Effet de la taille des instances sur les performances des LLM pour le problème ULS 85

B.1 Instances p-median utilisées dans les expérimentations 102
B.2 Instances du *Set Covering Problem* utilisées dans les expérimentations . 103
B.3 Instances ULS utilisées dans les expérimentations 104

Liste des abréviations

ESSAT — École Supérieure en Sciences Appliquées de Tlemcen

Chapitre 1

IA — Intelligence Artificielle

AI — Artificial Intelligence (voir IA)

ML — Machine Learning (Apprentissage automatique)

DL — Deep Learning (Apprentissage profond)

LLM — Large Language Model (Grand modèle de langage)

NLP — Natural Language Processing (Traitement automatique du langage naturel)

GenAI — Generative Artificial Intelligence (IA générative)

GPT — Generative Pre-trained Transformer

RAG — Retrieval-Augmented Generation (Génération augmentée par recherche)

RL — Reinforcement Learning (Apprentissage par renforcement)

RO — Recherche Opérationnelle

MILP — Mixed Integer Linear Programming (Programmation linéaire en nombres entiers mixtes)

ILP — Integer Linear Programming (Programmation linéaire en nombres entiers)

LP — Linear Programming (Programmation linéaire)

TSP — Travelling Salesman Problem (Problème du voyageur de commerce)

VRP — Vehicle Routing Problem (Problème de tournées de véhicules)

CNN — Convolutional Neural Network (Réseau de neurones convolutif)

RNN — Recurrent Neural Network (Réseau de neurones récurrent)

LSTM — Long Short-Term Memory (Mémoire à long et court terme)

GRU — Gated Recurrent Unit (Unité récurrente à portes)

GNN — Graph Neural Network (Réseau de neurones sur graphe)

GAN — Generative Adversarial Network (Réseau génératif antagoniste)

VAE — Variational Autoencoder (Autoencodeur variationnel)

SVM — Support Vector Machine (Machine à vecteurs de support)

PCA — Principal Component Analysis (Analyse en composantes principales)

FAISS — Facebook AI Similarity Search

RLHF — Reinforcement Learning from Human Feedback (Apprentissage par renforcement à partir de retours humains)

DPO — Direct Preference Optimization (Optimisation directe des préférences)

PPO — Proximal Policy Optimization (Optimisation de politique proximale)

ALNS — Adaptive Large Neighborhood Search (Recherche adaptative à grand voisinage)

NP — Non-deterministic Polynomial time (classe de complexité)

GPU — Graphics Processing Unit (Unité de traitement graphique)

TPU — Tensor Processing Unit (Unité de traitement tensoriel)

API — Application Programming Interface (Interface de programmation applicative)

JSON — JavaScript Object Notation (format de données structurées)

Chapitre 2

APO — Automatic Prompt Optimization (Optimisation automatique de prompts)

BLEU — Bilingual Evaluation Understudy (métrique d'évaluation de texte généré)

CoT — Chain-of-Thought (Chaîne de raisonnement)

PE — Prompt Engineering (Ingénierie des prompts)

RE — Requirements Engineering (Ingénierie des exigences)

ReAct — Reasoning and Acting (Raisonnement et action)

ROUGE — Recall-Oriented Understudy for Gisting Evaluation (métrique d'évaluation de résumé)

ToT — Tree of Thoughts (Arbre de raisonnements)

YAML — YAML Ain't Markup Language (format de sérialisation de données)

Chapitre 3

CoT — Chain-of-Thought (voir Chapitre 2)

OR-Library — Operations Research Library (bibliothèque de benchmarks en RO)

ULS — Uncapacitated Lot Sizing (dimensionnement de lots sans contrainte de capacité)

Chapitre 4

LLM — Large Language Model (voir Chapitre 1)

UPMP — Uncapacitated P-Median Problem (problème de la p-médiane non capacitée)

SCP — Set Covering Problem (problème de couverture par ensembles)

ULS — Uncapacitated Lot Sizing Problem (voir Chapitre 3)

GAP — Écart relatif entre la solution proposée par le LLM et la solution optimale ou de référence

OR-Library — Operations Research Library (voir Chapitre 3)

API — Application Programming Interface (voir Chapitre 1)
CPLEX — Solveur d'optimisation développé par IBM
IBM — International Business Machines
Gurobi — Solveur d'optimisation mathématique
OR-Tools — Suite d'outils d'optimisation développée par Google
MILP — Mixed Integer Linear Programming (voir Chapitre 1)
ILP — Integer Linear Programming (voir Chapitre 1)
LP — Linear Programming (voir Chapitre 1)
Nb. — Nombre
s — Seconde(s), unité utilisée pour les temps de réponse

Introduction générale

L'intelligence artificielle (IA) connaît aujourd'hui une évolution majeure, portée par les progrès du machine learning, du deep learning et, plus récemment, par l'émergence de l'intelligence artificielle générative. Cette dernière regroupe des systèmes capables de produire du contenu nouveau, comme du texte, du code, des images ou des raisonnements structurés, à partir d'instructions formulées en langage naturel. Elle transforme ainsi la manière dont les utilisateurs interagissent avec les outils numériques, en rendant l'accès à des systèmes d'IA avancés plus direct, plus flexible et plus intuitif.

Parmi les technologies les plus représentatives de cette évolution, les grands modèles de langage, ou *Large Language Models* (LLM), occupent une place centrale. Des modèles tels que ChatGPT, Gemini, DeepSeek, Grok ou Claude sont capables de comprendre des consignes complexes, de générer du code, de reformuler des problèmes, de produire des explications détaillées et d'assister les utilisateurs dans des tâches variées. Leur particularité est de permettre une interaction en langage naturel, sans exiger nécessairement de compétences avancées en programmation.

Cette accessibilité ouvre de nouvelles perspectives pour les domaines techniques et industriels. Les LLM ne sont plus utilisés uniquement pour la rédaction ou la génération de texte, mais aussi pour des tâches de raisonnement, de modélisation, de programmation et d'aide à la décision. Dans le contexte du génie industriel, ces capacités sont particulièrement intéressantes, car elles pourraient aider des utilisateurs non experts en informatique ou en recherche opérationnelle à traiter des problèmes complexes.

Cependant, malgré leurs performances impressionnantes, les LLM présentent encore des limites importantes. Ils peuvent produire des réponses convaincantes mais incorrectes, commettre des erreurs de calcul, oublier des contraintes, générer du code non fonctionnel ou affirmer des résultats non vérifiés. Cette difficulté est particulièrement critique dans les domaines où les réponses doivent être exactes, vérifiables et exploitables, comme l'ingénierie, la planification, la logistique ou l'optimisation.

C'est dans ce contexte que s'inscrit ce mémoire d'ingéniorat. L'objectif général est d'étudier le potentiel réel de l'intelligence artificielle générative, et plus précisément des LLM, dans un cadre technique nécessitant du raisonnement structuré et le respect de contraintes. Il ne s'agit pas seulement d'observer si ces modèles peuvent produire une réponse, mais de vérifier si cette réponse est fiable, cohérente, faisable et utile dans une démarche d'aide à la décision.

Pour cela, l'optimisation combinatoire est utilisée comme domaine d'application. Ce choix se justifie par le fait que les problèmes d'optimisation sont fréquents en génie industriel, notamment dans la localisation d'installations, la couverture de demandes, la planification de production et l'affectation de ressources. Ils constituent également un cadre d'évaluation rigoureux, car les solutions proposées peuvent être vérifiées ob-

jectivement à travers des critères comme la faisabilité, la valeur de la fonction objectif et l'écart par rapport à une solution de référence.

La problématique générale de ce travail peut ainsi être formulée comme suit :

> *Dans quelle mesure les modèles d'intelligence artificielle générative, et plus particulièrement les grands modèles de langage, peuvent-ils contribuer à la résolution de problèmes techniques nécessitant du raisonnement structuré, et quelles sont les limites de leur utilisation comme outils autonomes d'aide à la décision ?*

Cette problématique place l'IA générative au centre du travail, tandis que l'optimisation combinatoire constitue le terrain expérimental permettant d'évaluer concrètement les capacités et les limites des modèles étudiés.

Pour répondre à cette problématique, ce mémoire adopte une démarche progressive. Le premier chapitre présente les fondements de l'intelligence artificielle, du machine learning, du deep learning, de l'intelligence artificielle générative et des LLM. Il permet de situer les LLM dans l'évolution globale de l'IA et de comprendre leurs capacités, mais aussi leurs limites, notamment lorsqu'ils sont utilisés dans des tâches nécessitant un raisonnement formel ou une vérification externe.

Le deuxième chapitre est consacré à l'état de l'art du *prompt engineering*. Il présente les principales techniques de prompting, telles que le *zero-shot prompting*, le *few-shot prompting*, le *Chain-of-Thought*, le *role prompting*, le *format prompting* et le *negative prompting*. Ce chapitre montre que la qualité des réponses produites par un LLM dépend fortement de la manière dont la demande lui est formulée.

Le troisième chapitre propose un framework de *prompt engineering* adapté à l'optimisation combinatoire. Ce framework vise à construire des prompts structurés, comparables et reproductibles. Il repose notamment sur l'utilisation de l'anglais, l'attribution d'un rôle d'expert, la décomposition de la tâche, la spécification du format de sortie et l'interdiction de reproduire directement des solutions optimales connues.

Le quatrième chapitre présente les résultats expérimentaux obtenus lors de la première phase d'évaluation. Cinq LLM sont testés : **ChatGPT 5.3**, **Gemini 3**, **DeepSeek**, **Grok 4** et **Claude Sonnet 4.6**. Ces modèles sont évalués comme solveurs autonomes sur trois problèmes classiques d'optimisation combinatoire : le p-median, le *Set Covering Problem* et l'*Uncapacitated Lot Sizing Problem*. Les réponses produites sont analysées selon plusieurs indicateurs, notamment la qualité de la solution, la faisabilité, la fiabilité, la stabilité, le temps de réponse et la capacité d'amélioration au fil des interactions.

À travers cette démarche, ce mémoire vise à contribuer à la compréhension du rôle que peut jouer l'intelligence artificielle générative dans les domaines techniques. L'objectif n'est pas de présenter les LLM comme des substituts directs aux méthodes classiques, mais d'évaluer leur potentiel comme outils d'assistance, de modélisation, de génération de solutions et d'aide à la décision. La distinction entre une réponse bien formulée et une solution réellement valide constitue ainsi un point central de ce travail.

Chapitre 1

Intelligence artificielle : concepts, couches technologiques et émergence des LLM

1.1 Introduction générale

L'intelligence artificielle (IA) s'est imposée comme une composante majeure des systèmes numériques modernes, avec une présence croissante dans les processus industriels, la logistique, l'ingénierie et les services [11]. Son ambition est d'apporter une assistance à la décision, une automatisation partielle ou complète de tâches, ainsi qu'une meilleure exploitation des données massives.

L'IA moderne a connu plusieurs phases d'essor et de désillusion depuis les années 1950. La renaissance du deep learning autour de 2012, marquée par la victoire d'AlexNet sur ImageNet, a ouvert une nouvelle ère. L'arrivée des Transformers en 2017, introduits dans l'article fondateur *Attention is all you need*, a constitué un tournant décisif. Cette architecture a conduit directement au développement des grands modèles de langage, ou *Large Language Models* (LLM), tels que ChatGPT, Gemini, Claude, DeepSeek ou Grok [6].

Il existe une différence fondamentale entre **prédire** et **décider**. Prédire consiste à estimer une valeur ou un état futur, comme une demande, un délai ou un risque. Décider consiste à choisir une action parmi un ensemble de possibilités, en respectant des contraintes et en optimisant un objectif. Dans l'industrie, de nombreuses tâches appartiennent au second type : planification de production, allocation de ressources, ordonnancement ou tournées de livraison. Ces problèmes relèvent souvent de l'**optimisation combinatoire**, où l'espace de solutions est discret et augmente très rapidement avec la taille du problème [8].

L'émergence des LLM ouvre une question centrale pour ce mémoire : **quel est le potentiel réel des LLM pour résoudre des problèmes d'optimisation combinatoire ?** Sont-ils capables de produire des solutions faisables ? Peuvent-ils proposer des heuristiques efficaces ? Sont-ils plus utiles comme assistants de modélisation, comme générateurs de solutions initiales, ou comme outils d'aide à la formulation et à la vérification de solutions [29] ?

Cette question prend d'autant plus d'importance que la littérature récente insiste sur la nécessité d'évaluer rigoureusement les approches apprenantes appliquées à l'optimisation

combinatoire [8]. D'autres travaux cherchent à rendre les LLM plus fiables en les ancrant dans des signaux formels via des solveurs [5]. L'intégration de l'IA générative en recherche opérationnelle est synthétisée dans des revues récentes [29], et l'usage des LLM en optimisation métaheuristique est analysé de manière structurée [12].

Plan du chapitre

Ce chapitre adopte une progression en couches, du plus général vers le plus spécifique :

- Clarification des concepts : IA, ML, DL, IA générative et LLM ;
- Approches historiques et modernes de l'IA et leurs limites ;
- Fondations du machine learning [11] ;
- Le deep learning : architectures et mécanismes fondateurs ;
- Les LLM : fonctionnement, capacités émergentes et limites critiques [6] ;
- Bonnes pratiques expérimentales en machine learning appliqué ;
- Enjeux de fiabilité, d'évaluation et de reproductibilité pour l'optimisation combinatoire [8].

1.2 Clarification des couches : IA, ML, DL, IA générative et LLM

De nombreux termes sont utilisés de manière interchangeable dans les conversations courantes : IA, machine learning, deep learning, IA générative ou LLM. Pourtant, comme le souligne [6], ces notions n'appartiennent pas au même niveau. Certaines sont des **domaines**, comme l'IA, tandis que d'autres sont des **familles de méthodes** ou des **modèles spécialisés**, comme le ML, le DL et les LLM. La Figure 1.1 illustre cette hiérarchie.

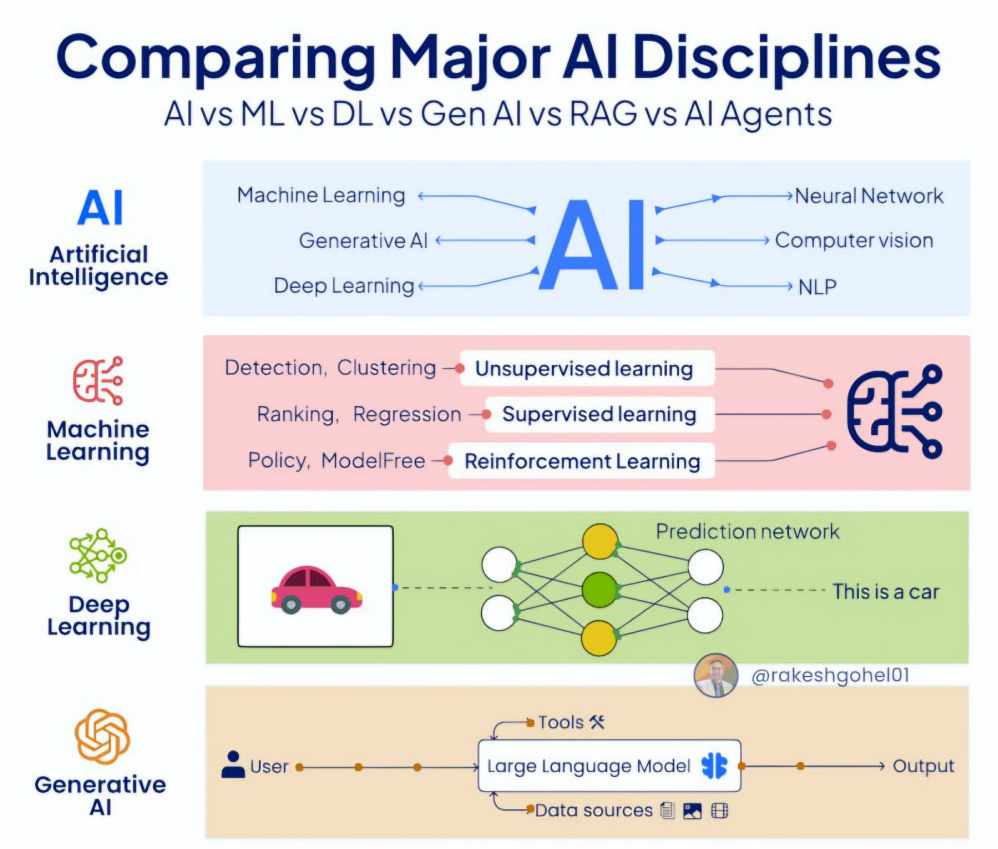


Figure 1.1: Hiérarchie des notions : IA, ML, DL, IA générative et LLM.

1.2.1 L'IA : la notion « parapluie »

L'intelligence artificielle est un ensemble large d'approches visant à doter les machines de capacités associées à l'intelligence humaine : percevoir, comprendre, apprendre, raisonner, planifier et décider [6]. À ce niveau, dire « IA » ne décrit pas une technique précise. Il s'agit plutôt d'un cadre général qui englobe des approches très hétérogènes, depuis les systèmes experts des années 1980 jusqu'aux réseaux neuronaux profonds actuels.

Dans le contexte industriel, l'IA peut intervenir à plusieurs niveaux : prédiction de la demande, détection d'anomalies, aide à la planification, analyse de données de production ou assistance à la décision. Elle ne se limite donc pas à un seul type d'algorithme, mais constitue un ensemble de méthodes et d'outils pouvant être mobilisés selon la nature du problème étudié.

1.2.2 Machine learning : apprendre à partir de données

Le machine learning (ML) est un sous-ensemble de l'IA dont l'idée centrale est d'éviter d'écrire des règles explicites pour chaque situation, et de laisser le modèle apprendre des régularités à partir de données [11]. Le schéma de base est :

$$\text{Données} \rightarrow \text{Algorithme d'apprentissage} \rightarrow \text{Modèle} \rightarrow \text{Prédiction / Décision}$$

Comme le détaille [11], le ML se décompose classiquement en trois paradigmes :

- **Apprentissage supervisé** : le modèle apprend une correspondance entre des entrées et des sorties étiquetées. Exemples : régression linéaire, forêts aléatoires, SVM, réseaux neuronaux.
- **Apprentissage non supervisé** : le modèle cherche des structures dans des données non étiquetées. Exemples : k-means, DBSCAN, autoencodeurs.
- **Apprentissage par renforcement** : un système apprend par interaction avec un environnement en maximisant une récompense cumulée. Exemples : DQN, PPO, AlphaGo.

Le ML est particulièrement utile lorsqu'il existe des données historiques suffisantes pour apprendre des relations entre variables. En revanche, il ne garantit pas nécessairement le respect de contraintes strictes, ce qui constitue une limite importante dans les problèmes d'optimisation combinatoire.

1.2.3 Deep learning : réseaux neuronaux profonds

Le deep learning (DL) est une sous-catégorie du ML basée sur des réseaux de neurones artificiels composés de nombreuses couches [11]. Il apprend automatiquement des représentations hiérarchiques adaptées aux données, sans ingénierie manuelle de caractéristiques. Cette capacité a été déterminante dans le traitement automatique du langage, en améliorant fortement la capacité des modèles à capturer le contexte et les dépendances longues.

Le deep learning a connu une forte progression grâce à trois facteurs principaux : l'augmentation massive des données disponibles, l'amélioration des architectures neuronales et la disponibilité de ressources de calcul plus puissantes. Ces progrès ont permis l'émergence des Transformers, puis des grands modèles de langage.

1.2.4 IA générative : produire du contenu nouveau

L'IA générative regroupe les modèles capables de créer du contenu nouveau : texte, code, images, audio ou vidéo [6]. Les LLM appartiennent à cette catégorie lorsqu'ils génèrent une séquence de texte ou de code. Comme le soulignent [29], l'usage de la GenAI en recherche opérationnelle ouvre des perspectives importantes pour la modélisation, l'explication et la résolution de problèmes complexes.

Dans ce mémoire, l'intérêt principal porte sur la capacité des LLM à générer des réponses structurées dans un contexte technique : formulation mathématique, proposition de solutions, génération de code, explication des étapes de résolution et justification des résultats.

1.2.5 LLM : modèles de langage à grande échelle

Les grands modèles de langage, ou LLM, sont des modèles génératifs entraînés sur de très grands corpus textuels afin de prédire et générer des séquences de tokens [6].

Leur force réside dans leur capacité à produire des réponses cohérentes, à suivre des consignes, à générer du code et à reformuler des problèmes complexes.

Cependant, un LLM ne raisonne pas comme un solveur mathématique classique. Il génère des réponses probables à partir de son apprentissage et du contexte fourni. Cela signifie qu'il peut proposer des solutions plausibles mais incorrectes, notamment lorsque les contraintes sont nombreuses ou lorsque des calculs précis sont nécessaires. Cette distinction est fondamentale pour l'évaluation menée dans ce mémoire.

1.2.6 Synthèse hiérarchique

Table 1.1: Hiérarchie des notions IA et pertinence pour l'optimisation combinatoire

Couche	**Nature**	**Exemples**	**Pertinence pour l'optimisation**
IA	Domaine général	Systèmes experts, ML, DL	Cadre général de l'automatisation, de l'aide à la décision et de la résolution de problèmes complexes.
ML	Famille de méthodes	SVM, forêts, régression	Prédictions en amont de l'optimisation, par exemple la demande, les délais ou les risques.
DL	Sous-famille du ML	CNN, LSTM, Transformer	Représentations complexes et traitement de données non structurées.
IA générative	Capacité générative	ChatGPT, Gemini, Claude	Génération de texte, de code, de formulations et de solutions candidates.
LLM	Modèles génératifs de langage	ChatGPT 5.3, Gemini 3, DeepSeek, Grok 4, Claude Sonnet 4.6	Raisonnement textuel, formulation mathématique et assistance à la résolution de problèmes d'optimisation combinatoire.

1.3 Approches de l'IA : symbolique, statistique et hybridation

1.3.1 IA symbolique

L'IA symbolique manipule des représentations explicites : règles, logique formelle, contraintes, graphes, ontologies ou systèmes experts. Son avantage principal est la **traçabilité**. On peut expliquer une décision en retraçant exactement les règles ou contraintes appliquées. Une solution produite par un solveur MILP est garantie faisable ou optimale selon le modèle défini [8].

Ses limites sont également importantes : difficulté à capturer la complexité du monde réel, sensibilité aux exceptions et nécessité d'un expert humain pour définir les règles.

Dans des environnements industriels dynamiques, ces limites peuvent rendre les systèmes purement symboliques difficiles à maintenir.

1.3.2 IA statistique ou connexionniste

L'IA statistique repose sur l'apprentissage à partir de données [11]. Elle est généralement plus flexible et plus robuste à la variabilité, mais elle dépend fortement des données utilisées : biais, représentativité, bruit et qualité d'étiquetage. Avec des modèles profonds, il devient également difficile d'expliquer exactement pourquoi une sortie a été produite.

Cette approche a permis des progrès importants en vision artificielle, en traitement du langage naturel et en prédiction. Cependant, dans les problèmes à contraintes, une bonne prédiction ne suffit pas toujours à produire une décision valide.

1.3.3 Hybridation : une approche naturelle pour des tâches à contraintes

En pratique, les systèmes les plus adaptés aux problèmes industriels complexes sont souvent hybrides [5]. En optimisation combinatoire, cette hybridation est presque inévitable : la génération d'une solution candidate doit être suivie d'une vérification des contraintes et d'un calcul objectif. Plusieurs directions peuvent être distinguées :

- **LLM + solveur exact** : le LLM peut aider à générer une formulation MILP ou CP, tandis que le solveur assure la résolution et la vérification formelle de la faisabilité [5].
- **LLM + heuristique** : le LLM peut proposer une solution initiale ou une stratégie de résolution, qui peut ensuite être améliorée par une méthode heuristique ou métaheuristique.
- **ML + optimisation** : le modèle prédictif estime certains paramètres, comme la demande ou les délais, puis un modèle d'optimisation prend la décision finale sous contraintes [8].

Cette logique hybride est centrale dans ce mémoire. Les LLM sont évalués non seulement sur leur capacité à produire une réponse, mais surtout sur la validité réelle des solutions proposées après vérification.

1.4 Machine learning : fondations utiles pour l'optimisation

1.4.1 Apprentissage supervisé

L'apprentissage supervisé est la branche la plus utilisée du ML en contexte industriel [11]. Il apprend à prédire une sortie à partir d'entrées, en minimisant une fonction de perte sur un ensemble de données étiquetées. Parmi les algorithmes classiques, on peut citer :

- la régression linéaire et logistique : modèles simples et interprétables ;
- les arbres de décision et forêts aléatoires : robustes et efficaces sur des données tabulaires ;
- le gradient boosting, comme XGBoost ou LightGBM : souvent performant sur données structurées ;
- les Support Vector Machines (SVM) : efficaces pour des espaces de grande dimension ;
- les réseaux de neurones : flexibles, scalables et adaptés aux données non structurées.

Dans une chaîne décisionnelle logistique ou industrielle, le ML supervisé est souvent un outil *amont* de l'optimisation : prédiction de la demande par ML, puis optimisation du plan de production à partir de ces prévisions [11].

1.4.2 Apprentissage non supervisé

L'apprentissage non supervisé vise à identifier des structures dans des données sans étiquettes [11]. En optimisation, il peut servir à segmenter des instances ou à identifier différents régimes de fonctionnement :

- **Clustering** : segmentation de zones géographiques pour un problème de routage ;
- **Réduction de dimension** : compression pour la visualisation ou l'analyse exploratoire ;
- **Détection d'anomalies** : maintenance prédictive ou contrôle qualité.

Ces méthodes ne résolvent pas directement un problème d'optimisation, mais elles peuvent améliorer la qualité des données utilisées dans une démarche décisionnelle.

1.4.3 Apprentissage par renforcement

L'apprentissage par renforcement apprend par interaction avec un environnement via une récompense. Il est particulièrement pertinent pour l'optimisation combinatoire séquentielle : construire une tournée client par client, affecter des tâches une à une ou sélectionner progressivement des décisions. Dans le contexte des LLM, certaines approches utilisent un solveur comme signal pour guider le comportement du modèle, en récompensant la validité ou la qualité de la solution [5].

Cette approche reste cependant complexe à mettre en œuvre. Elle nécessite une définition précise de l'environnement, de l'espace d'actions, de la récompense et des critères d'arrêt. Elle est donc plus adaptée à des cadres expérimentaux contrôlés qu'à une utilisation directe sans vérification.

1.4.4 Limites du ML en présence de contraintes

Le ML excelle à prédire des tendances moyennes, mais ne garantit pas le respect strict de contraintes [11]. Une prédiction plausible peut violer une règle métier. C'est pourquoi, pour des décisions sous contraintes, on combine souvent ML et méthodes de recherche opérationnelle : le ML propose ou estime, et l'optimisation décide sous contraintes [8].

Cette limite est encore plus importante pour les LLM, car une réponse textuellement cohérente peut cacher une solution infaisable ou une valeur objective incorrecte.

1.5 Deep learning : pourquoi il a ouvert la voie aux LLM

1.5.1 Le neurone artificiel et le perceptron

Un réseau de neurones artificiel est inspiré du cerveau biologique de manière abstraite [11]. Le neurone artificiel de base reçoit des entrées $x_1, x_2, \ldots, x_n$, les pondère avec des poids apprenables $w_1, w_2, \ldots, w_n$, calcule la somme pondérée :

$$z = \sum_{i=1}^{n} w_i x_i + b$$

et applique une fonction d'activation f, comme sigmoïde, ReLU ou tanh, pour produire une sortie $a = f(z)$.

Un réseau profond empile de nombreuses couches. La propagation avant (*forward pass*) calcule la prédiction, et la rétropropagation (*backpropagation*) ajuste les poids en calculant le gradient de la perte via la règle de la chaîne [11].

1.5.2 Architectures clés et leur pertinence

Plusieurs architectures ont marqué l'histoire du deep learning [11] :

- **CNN** (*Convolutional Neural Networks*) : exploitent la localité et l'invariance par translation dans les images. Ils ont révolutionné la vision artificielle, notamment avec AlexNet en 2012 et ResNet en 2015.
- **RNN / LSTM / GRU** : conçus pour les séquences temporelles. Les LSTM gèrent mieux les dépendances longues grâce à des portes de contrôle. Ces architectures ont dominé le NLP avant les Transformers.
- **Transformers** : introduits par Vaswani et al. en 2017, basés sur l'attention multi-têtes. Ils sont aujourd'hui dominants en NLP, en vision et en multimodal [6].
- **Graph Neural Networks (GNN)** : opèrent sur des structures de graphes et peuvent être utilisés pour représenter des réseaux, des relations ou des structures combinatoires.

1.5.3 Le mécanisme d'attention : fondateur des LLM

Le mécanisme d'attention calcule une pondération dynamique entre tous les tokens d'une séquence [6]. Chaque token est projeté en trois vecteurs : Query (Q), Key (K) et Value (V). L'attention est calculée comme suit :

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Ce mécanisme permet [6] :

- une vision globale de la séquence, contrairement aux RNN séquentiels ;
- la modélisation des dépendances à longue portée ;
- la parallélisation des calculs, rendant l'entraînement beaucoup plus rapide.

L'attention multi-têtes (*Multi-Head Attention*) exécute plusieurs mécanismes d'attention en parallèle, capturant simultanément différents types de relations syntaxiques, sémantiques ou pragmatiques.

1.5.4 Architecture Transformer et encodage positionnel

Les Transformers ne traitent pas les tokens de manière séquentielle. Un encodage positionnel est donc ajouté aux embeddings d'entrée. L'encodage original utilise des fonctions sinusoïdales, comme proposé par Vaswani et al. en 2017. Les LLM actuels de type GPT sont des Transformers décodeurs uniquement (*decoder-only*), qui génèrent le texte token par token [6].

Cette architecture explique en partie la capacité des LLM à produire du texte cohérent, à maintenir un contexte sur plusieurs phrases et à générer des raisonnements structurés. Cependant, elle n'implique pas une garantie de calcul exact ou de validité logique.

1.5.5 Coût, robustesse et interprétabilité

Le deep learning nécessite des ressources importantes : grandes quantités de données, puissance de calcul, GPU ou TPU, et temps d'entraînement élevé [11]. L'interprétabilité reste limitée : il est difficile d'expliquer précisément pourquoi une sortie a été produite. Pour l'optimisation combinatoire, cette limite est particulièrement importante, car une solution doit être valide, vérifiable et souvent explicable pour être acceptée par les acteurs métier [8].

1.6 Grands modèles de langage (LLM) : fonctionnement, capacités et limites

1.6.1 Définition et notion de tokens

Un LLM est un Transformer de très grande taille, allant de quelques milliards à un très grand nombre de paramètres, entraîné à prédire le prochain token d'une séquence [6].

Un token n'est pas forcément un mot entier : il peut être un sous-mot, un symbole, un espace ou un nombre. Le pré-entraînement se fait sur de vastes corpus : web, livres, code ou publications scientifiques. À grande échelle, cette tâche induit des **capacités émergentes**, comme suivre des consignes, reformuler, expliquer ou générer du code [6].

1.6.2 Étapes d'entraînement des LLM

[6] distingue plusieurs phases dans l'entraînement d'un LLM moderne :

- **Pré-entraînement** : apprentissage général sur un vaste corpus. L'objectif est la prédiction du prochain token. C'est la phase la plus coûteuse en calcul.
- **Instruction tuning** : adaptation du modèle pour suivre des consignes via des paires question-réponse supervisées.
- **RLHF** (*Reinforcement Learning from Human Feedback*) : un modèle de récompense est entraîné à partir d'annotations humaines, puis utilisé dans une boucle de renforcement, souvent avec PPO.
- **DPO** (*Direct Preference Optimization*) : alternative plus stable au RLHF, optimisant directement selon des préférences comparatives.

Ces étapes visent à rendre le modèle plus utile, plus cohérent et plus aligné avec les attentes humaines. Elles ne transforment cependant pas le LLM en solveur exact : une réponse bien formulée doit toujours être vérifiée lorsqu'elle concerne un problème à contraintes.

1.6.3 Capacités émergentes des LLM

[6] et [29] identifient plusieurs capacités émergentes des LLM particulièrement utiles en contexte applicatif :

- **Raisonnement en chaîne** (*chain-of-thought*) : en invitant le modèle à structurer son raisonnement, on améliore ses performances sur des tâches multi-étapes.
- **Few-shot learning** : en fournissant quelques exemples dans le prompt, le modèle généralise sans réentraînement.
- **Génération de code** : écriture, débogage et explication dans plusieurs langages de programmation.
- **Reformulation de problèmes** : passage d'un énoncé en langage naturel à une formulation structurée ou mathématique [24].

Ces capacités expliquent pourquoi les LLM peuvent être utiles dans des contextes techniques. Elles doivent toutefois être distinguées d'une capacité à garantir l'optimalité ou la faisabilité d'une solution.

1.6.4 Capacités utiles en optimisation combinatoire

[29] et [12] identifient plusieurs niveaux d'utilisation des LLM pour l'optimisation :

Table 1.2: Capacités des LLM applicables à l'optimisation combinatoire [29]

Capacité	**Description**	**Exemple concret**
Structuration	Extraire les contraintes et l'objectif	Problème de planification décrit textuellement [24]
Modélisation	Formuler en MILP ou CP	Modèle mathématique pour un problème de production
Heuristiques	Proposer une stratégie de résolution	Algorithme glouton pour un problème de couverture
Code	Générer un script exécutable et vérifiable	Vérificateur Python pour contrôler la faisabilité [5]
Explication	Fournir un raisonnement traçable	Expliquer une violation de contrainte
Comparaison	Discuter plusieurs approches possibles	Méthode exacte, heuristique ou métaheuristique

1.6.5 Paramètres de génération et stabilité

La génération d'un LLM dépend de plusieurs paramètres de décodage [6] :

- **Température** : contrôle l'aléatoire. Une température faible produit des réponses plus déterministes, ce qui est utile pour la reproductibilité.
- **Top-p** (*nucleus sampling*) : restreint la génération à une distribution cumulée de probabilité.
- **Top-k** : restreint la génération aux k tokens les plus probables.
- **Max tokens** : limite la longueur de la génération.

En optimisation combinatoire, la stabilité est un enjeu majeur [8]. Cela justifie des protocoles avec répétitions, comparaison des réponses et vérification externe.

1.6.6 Limites critiques des LLM en présence de contraintes

[8], [5] et [12] convergent sur les limites suivantes :

- **Hallucination** : réponse plausible mais factuellement incorrecte. En optimisation, cela peut se traduire par une solution qui semble logique mais viole des contraintes.

- **Oubli de contraintes** : même avec un prompt complet, le LLM peut ne pas intégrer toutes les contraintes, surtout lorsqu'elles sont nombreuses ou complexes.
- **Erreurs de calcul** : les LLM ne sont pas des calculatrices. Des erreurs arithmétiques et des confusions d'indices sont fréquentes.
- **Sensibilité au prompt** : une formulation légèrement différente peut produire des résultats très différents [26].
- **Absence de garanties** : le modèle ne fournit pas de preuve d'optimalité ni de garantie formelle de faisabilité.
- **Fenêtre de contexte limitée** : les problèmes à grande échelle peuvent dépasser les capacités d'un seul prompt.

Ces limites motivent l'utilisation de protocoles de vérification externes, notamment à travers des solveurs, des scripts de validation et le recalcul systématique des fonctions objectifs [5].

1.7 Conclusion

Ce chapitre a posé les fondations conceptuelles et techniques nécessaires à la compréhension de nos travaux. Nous avons progressivement clarifié la hiérarchie des notions, de l'IA vers le ML et le DL, puis montré comment les LLM s'insèrent dans cet écosystème [6].

L'analyse critique de leurs capacités et de leurs limites en contexte d'optimisation combinatoire est au cœur de notre positionnement [8, 5]. Les LLM peuvent structurer des problèmes, générer du code, proposer des heuristiques et expliquer des raisonnements [29]. Cependant, ils ne peuvent pas garantir seuls la faisabilité ou l'optimalité, commettent des erreurs de calcul et sont sensibles à la formulation du prompt [12].

La voie la plus prometteuse repose sur l'**hybridation** [5] : un LLM peut servir de générateur d'idées, de formulations ou de code, tandis que les solveurs et les scripts de vérification apportent une validation formelle des solutions proposées.

Ces éléments théoriques justifient le choix d'évaluer expérimentalement plusieurs modèles récents, notamment ChatGPT 5.3, Gemini 3, DeepSeek, Grok 4 et Claude Sonnet 4.6, sur des problèmes d'optimisation combinatoire vérifiables.

Le chapitre suivant présentera l'état de l'art du **prompt engineering** : ses techniques, ses patterns et son rôle central dans l'utilisation efficace des LLM pour l'optimisation combinatoire.

Chapitre 2

Prompt Engineering : techniques, cadres et applications

2.1 Introduction

L'émergence des grands modèles de langage (*Large Language Models*, LLM) a profondément transformé la manière dont les systèmes d'intelligence artificielle interagissent avec leurs utilisateurs. Contrairement aux modèles classiques d'apprentissage automatique, qui nécessitent généralement un réentraînement ou un *fine-tuning* pour s'adapter à de nouvelles tâches, les LLM peuvent adapter leur comportement à partir des instructions qui leur sont fournies en entrée. Cette instruction, appelée **prompt**, peut prendre la forme d'une question, d'une consigne, d'un contexte, d'un exemple ou d'une combinaison de ces éléments.

Le *prompt engineering* désigne l'ensemble des méthodes visant à concevoir, structurer et améliorer les prompts afin d'obtenir des réponses plus pertinentes, plus fiables et mieux adaptées à une tâche donnée [9, 6]. Cette discipline repose sur une observation centrale : pour un même modèle et une même tâche, deux formulations différentes peuvent produire des réponses très différentes en termes de qualité, de précision, de cohérence ou de respect des contraintes.

Dans le contexte de ce mémoire, le *prompt engineering* occupe une place essentielle. L'évaluation comparative de plusieurs LLM, notamment **ChatGPT 5.3**, **Gemini 3**, **DeepSeek**, **Grok 4** et **Claude Sonnet 4.6**, sur des problèmes d'optimisation combinatoire repose sur la conception de prompts structurés, comparables et reproductibles. La qualité des solutions obtenues dépend donc non seulement du modèle utilisé, mais aussi de la manière dont la tâche lui est présentée.

Ce chapitre propose un état de l'art synthétique du *prompt engineering*. Il présente d'abord ses fondements conceptuels, puis l'anatomie d'un prompt, les principales techniques utilisées, les patterns de conception, les applications à l'optimisation mathématique, ainsi que les limites et défis ouverts.

2.2 Fondements conceptuels du *Prompt Engineering*

2.2.1 Définition et positionnement

Dans le contexte des LLM, un *prompt* désigne toute séquence d'entrée textuelle fournie au modèle pour guider sa génération. Il peut s'agir d'une question directe, d'une instruction détaillée, d'un ensemble d'exemples, d'un contexte technique ou d'un format de sortie imposé. Le *prompt engineering* consiste alors à concevoir ces entrées de manière méthodique afin d'améliorer la qualité des réponses générées [9, 22].

Il convient de distinguer le *prompt engineering* du *fine-tuning*. Le *fine-tuning* modifie les paramètres internes du modèle à partir de données spécifiques, ce qui nécessite des ressources importantes et des données annotées. À l'inverse, le *prompt engineering* agit uniquement au moment de l'inférence, sans modifier le modèle. Il constitue donc une méthode flexible, légère et accessible pour adapter le comportement d'un LLM à une tâche donnée.

Dans un contexte d'optimisation combinatoire, cette distinction est importante. L'objectif n'est pas de réentraîner les modèles, mais de tester leur capacité à comprendre un problème, à le formuler, à proposer une solution et à expliquer leur raisonnement à partir d'un prompt soigneusement construit.

2.2.2 Sensibilité des LLM aux prompts

Les LLM génèrent leurs réponses en prédisant progressivement les tokens les plus probables à partir du contexte fourni. Le prompt influence donc directement la distribution de probabilité des tokens générés. Une modification apparemment mineure de la consigne peut changer la représentation interne du contexte et conduire à une réponse différente [25].

Cette sensibilité explique pourquoi le *prompt engineering* est devenu une étape essentielle dans l'utilisation des LLM. Elle constitue à la fois une opportunité et une limite. D'un côté, une consigne bien structurée peut améliorer la précision, la cohérence et la forme de la réponse. De l'autre, cette dépendance à la formulation complique la reproductibilité, en particulier dans des tâches techniques où les réponses doivent être vérifiables.

2.2.3 Évolution du domaine

Le *prompt engineering* s'est développé avec l'apparition des grands modèles de langage capables de réaliser des tâches en *zero-shot* ou *few-shot*. Les premières pratiques étaient largement empiriques : tester différentes formulations, observer les réponses et ajuster les consignes. Progressivement, la littérature a proposé des taxonomies, des patterns réutilisables et des cadres méthodologiques [26, 25].

Aujourd'hui, le *prompt engineering* est appliqué à des domaines spécialisés comme l'ingénierie des exigences [23], la génération de données structurées [7], l'ingénierie en général [22] et l'optimisation mathématique [24]. Cette spécialisation montre que les prompts efficaces dépendent fortement du domaine, du type de tâche et des critères d'évaluation.

2.3 Anatomie d'un prompt : composantes et structure

2.3.1 Composantes fondamentales

Un prompt bien conçu ne se limite pas à une simple question. Il combine généralement plusieurs composantes, dont chacune joue un rôle précis dans l'orientation de la réponse [6, 25].

Le rôle. Le rôle, ou *persona*, indique au modèle la posture à adopter. Par exemple, l'instruction *« You are an expert in combinatorial optimization and mathematical programming »* oriente le modèle vers un style plus technique et une réponse plus structurée.

Le contexte. Le contexte fournit les informations nécessaires à la compréhension de la tâche : définition du problème, données de l'instance, conventions de notation, contraintes particulières ou objectifs de l'étude.

L'instruction. L'instruction précise ce que le modèle doit produire. Elle doit être claire, explicite et non ambiguë. Dans un problème d'optimisation, elle peut demander de formuler le modèle mathématique, de calculer une solution, de fournir les variables de décision ou de justifier la méthode utilisée.

Les contraintes. Les contraintes indiquent ce que la réponse doit respecter ou éviter. Elles peuvent porter sur le contenu, le format, la méthode ou les erreurs à ne pas commettre. Dans ce mémoire, une contrainte importante consiste à interdire au LLM de réutiliser directement des solutions optimales connues des benchmarks.

Le format de sortie. Le format de sortie spécifie la manière dont la réponse doit être présentée : tableau, liste, code, formulation mathématique ou structure textuelle imposée. Cette composante est essentielle lorsque les résultats doivent être vérifiés ou comparés automatiquement [7].

2.3.2 Structure et ordonnancement des composantes

L'ordre des composantes influence la réponse générée. Les instructions placées au début et à la fin du prompt sont généralement mieux prises en compte que celles situées au milieu, phénomène souvent associé au biais de position dans les mécanismes d'attention [25].

Dans ce travail, les prompts suivent une structure stable afin de garantir la comparabilité entre les modèles évalués. Cette structure comprend généralement : le rôle attribué au modèle, le contexte du problème, les données de l'instance, les tâches à réaliser, les contraintes à respecter et le format de sortie attendu. Cette organisation réduit les ambiguïtés et facilite la vérification des réponses produites.

2.3.3 Paramètres de génération

La qualité de la réponse dépend aussi des paramètres de génération du modèle [6]. Les plus importants sont :

- **Température** : contrôle le degré d'aléatoire. Une température faible favorise des réponses plus déterministes, ce qui est préférable pour les tâches techniques.
- **Top-p** : limite la génération aux tokens dont la probabilité cumulée atteint un seuil donné.
- **Top-k** : restreint la génération aux k tokens les plus probables.
- **Longueur maximale** : limite le nombre de tokens générés.

Dans l'évaluation des LLM appliqués à l'optimisation combinatoire, ces paramètres doivent être fixés et documentés afin de garantir une comparaison plus équitable entre les modèles.

2.4 Taxonomie des techniques de *Prompt Engineering*

La littérature distingue plusieurs familles de techniques de *prompt engineering*, allant des approches simples aux méthodes de raisonnement structuré [22, 25].

2.4.1 Techniques de base

Zero-Shot Prompting

Le *zero-shot prompting* consiste à demander directement au modèle de réaliser une tâche sans fournir d'exemple. Le modèle s'appuie uniquement sur ses connaissances générales et sur la consigne donnée.

> *« Résous le problème du p-médian suivant et fournis les sites ouverts, les affectations et la valeur de la fonction objectif. »*

Cette approche est simple, mais elle reste fragile pour les problèmes nécessitant un raisonnement structuré. En optimisation combinatoire, elle peut conduire à des solutions infaisables, à des oublis de contraintes ou à des erreurs de calcul.

Few-Shot Prompting

Le *few-shot prompting* consiste à fournir quelques exemples de la tâche avant de demander au modèle de traiter une nouvelle instance [9]. Le modèle infère le schéma attendu à partir des exemples.

Cette technique est utile lorsque la tâche est complexe ou lorsque le format de réponse doit être strict. Elle peut améliorer la régularité des sorties, mais elle augmente la longueur du prompt et peut devenir coûteuse lorsque les instances sont volumineuses.

One-Shot Prompting

Le *one-shot prompting* est un cas particulier du *few-shot prompting*, avec un seul exemple. Il représente un compromis entre simplicité et guidage. Il est pertinent lorsque l'on veut montrer un format ou une démarche sans surcharger le prompt.

2.4.2 Techniques de raisonnement structuré

Chain-of-Thought Prompting

Le *Chain-of-Thought* (CoT) consiste à inviter le modèle à décomposer son raisonnement en étapes intermédiaires avant de produire la réponse finale [25, 22]. Cette technique est particulièrement utilisée pour les tâches arithmétiques, logiques ou multi-étapes.

Dans le contexte de l'optimisation combinatoire, le CoT peut aider le modèle à organiser sa démarche : comprendre les données, identifier les variables, formuler les contraintes, proposer une méthode, puis calculer une solution. Cependant, il ne garantit pas que le raisonnement produit soit correct. Un LLM peut générer une chaîne de raisonnement cohérente en apparence, mais contenant des erreurs mathématiques.

Zero-Shot Chain-of-Thought

Le *zero-shot CoT* ajoute simplement une consigne du type *« Let us think step by step »* à un prompt sans exemple. Cette méthode est facile à utiliser et peut améliorer les réponses sur des tâches nécessitant plusieurs étapes [25].

Dans ce mémoire, cette idée est reprise indirectement à travers des prompts qui demandent explicitement au modèle d'expliquer son raisonnement étape par étape, sans nécessairement imposer une chaîne de pensée complète.

Self-Consistency

La *self-consistency* consiste à générer plusieurs raisonnements indépendants puis à sélectionner la réponse la plus fréquente ou la plus cohérente [25]. Elle repose sur l'idée qu'une solution correcte peut être retrouvée par plusieurs chemins de raisonnement.

Cette technique est intéressante pour les problèmes où plusieurs démarches peuvent conduire au même résultat. Elle reste cependant coûteuse, car elle nécessite plusieurs générations pour une même instance.

Tree of Thoughts

Le *Tree of Thoughts* étend le *Chain-of-Thought* en explorant plusieurs branches de raisonnement au lieu d'une seule chaîne linéaire [25]. Le modèle peut générer plusieurs pistes, les évaluer et poursuivre celles qui semblent les plus prometteuses.

Cette logique se rapproche de certaines méthodes utilisées en optimisation, comme l'exploration arborescente. Toutefois, elle reste difficile à contrôler sans mécanisme externe de vérification.

2.4.3 Techniques de structuration et de contrôle

Role Prompting

Le *role prompting* consiste à assigner explicitement un rôle au modèle. White et al. [26] montrent que cette technique peut influencer le niveau de détail, le vocabulaire et la structure de la réponse.

Dans ce mémoire, le rôle d'expert en optimisation combinatoire est utilisé afin d'encourager des réponses plus rigoureuses, notamment dans la formulation mathématique et la justification des méthodes.

Instruction Prompting

L'*instruction prompting* repose sur des consignes détaillées et explicites. Il vise à réduire les ambiguïtés et à préciser les attentes concernant la tâche, le contenu et le format de la réponse [9].

Cette technique est centrale dans les prompts utilisés pour les problèmes étudiés, car les modèles doivent produire à la fois une solution, une explication, une méthode de résolution et des informations de performance.

Constrained / Format Prompting

Le *format prompting* impose une structure de sortie précise : tableau, liste, JSON, formulation LaTeX ou colonnes spécifiques. Elnashar et al. [7] montrent que la spécification explicite du format améliore la conformité des sorties structurées.

Dans les problèmes d'optimisation, ce point est crucial : une réponse bien structurée facilite le recalcul de la fonction objectif, l'identification des variables et le contrôle de la faisabilité.

Negative Prompting

Le *negative prompting* consiste à préciser ce que le modèle ne doit pas faire. Par exemple, dans un benchmark connu, il est possible d'interdire explicitement au modèle de réutiliser une solution optimale publiée. Cette technique permet de limiter certains comportements indésirables, comme l'oubli de contraintes ou la reproduction de réponses mémorisées [26].

2.4.4 Techniques avancées

Prompt Chaining

Le *prompt chaining* consiste à diviser une tâche complexe en plusieurs sous-tâches successives. La sortie d'un prompt sert alors d'entrée au suivant [25]. Pour un problème d'optimisation, une chaîne peut consister à :

1. identifier les données et paramètres ;

2. formuler les variables de décision ;
3. construire les contraintes ;
4. définir la fonction objectif ;
5. proposer une solution ;
6. vérifier la faisabilité.

Cette décomposition réduit la charge imposée au modèle à chaque étape. Elle est particulièrement pertinente pour les problèmes techniques, mais elle augmente le nombre d'interactions nécessaires.

Automatic Prompt Optimization

L'*Automatic Prompt Optimization* (APO) traite le prompt comme un objet à optimiser. Des variantes de prompts sont générées, testées puis sélectionnées selon une métrique de performance [22]. Cette approche peut améliorer les résultats, mais elle est coûteuse en appels au modèle et nécessite un ensemble d'évaluation bien défini.

Dans le cadre de ce mémoire, l'objectif n'est pas d'automatiser l'optimisation des prompts, mais de concevoir des prompts expérimentaux suffisamment structurés pour permettre une comparaison équitable entre LLM.

2.5 Catalogues de patterns et cadres méthodologiques

2.5.1 Le catalogue de patterns de White et al.

White et al. [26] proposent un catalogue de *prompt patterns*, inspiré des design patterns en génie logiciel. Ces patterns sont des solutions réutilisables à des problèmes récurrents de conception de prompts. Leur intérêt est de fournir un vocabulaire commun et des structures applicables dans différents contextes.

Table 2.1: Exemples de patterns de prompts utiles pour l'optimisation combinatoire

Pattern	**Principe**	**Intérêt pour l'optimisation**
Persona	Assigner un rôle au modèle	Orienter la réponse vers une expertise en recherche opérationnelle
Template	Imposer une structure de réponse	Faciliter la comparaison et la vérification automatique
Recipe	Demander une procédure étape par étape	Obtenir une méthode de résolution explicite
Fact Check List	Lister les affirmations à vérifier	Identifier les contraintes ou résultats à contrôler
Reflection	Demander au modèle de relire et corriger sa réponse	Réduire certaines erreurs de cohérence ou d'oubli
Alternative Approaches	Générer plusieurs méthodes possibles	Comparer méthode exacte, heuristique ou métaheuristique
Cognitive Verifier	Décomposer une tâche complexe en sous-questions	Structurer la formulation et la résolution d'un problème

Ces patterns sont particulièrement pertinents pour l'optimisation combinatoire, car ils permettent de mieux contrôler la réponse du modèle et de rendre son raisonnement plus exploitable.

2.5.2 Le framework Impromptu de Morales et al.

Morales et al. [21] proposent **Impromptu**, un framework *model-driven* pour la conception de prompts. L'idée principale est de considérer le prompt comme un artefact d'ingénierie, soumis à des exigences de clarté, de complétude, de maintenabilité et de réutilisabilité.

Le framework distingue plusieurs éléments : les composantes du prompt, les règles de composition, les métriques de qualité et les transformations possibles. Cette approche est intéressante pour des contextes expérimentaux où les prompts doivent être réutilisés sur plusieurs instances et plusieurs modèles.

Dans ce mémoire, cette logique inspire la nécessité de documenter les prompts, de conserver une structure stable et de rendre les consignes comparables entre les problèmes étudiés.

2.6 *Prompt Engineering* pour l'optimisation mathématique

L'application du *prompt engineering* à l'optimisation mathématique est directement liée au sujet de ce mémoire. Les travaux de Vu et al. [24] montrent que les LLM

peuvent aider à formuler des problèmes d'optimisation, mais qu'ils commettent aussi des erreurs fréquentes.

2.6.1 Erreurs de modélisation fréquentes

Vu et al. [24] identifient plusieurs catégories d'erreurs produites par les LLM lors de la formulation de problèmes d'optimisation :

- **Oubli de contraintes** : certaines contraintes présentes dans l'énoncé ne sont pas intégrées dans le modèle.
- **Mauvais sens d'inégalité** : une contrainte de type $\leq$ est transformée en $\geq$, ou inversement.
- **Erreur de domaine de variable** : une variable binaire est traitée comme continue, ou une variable positive est mal définie.
- **Confusion d'indices** : les sommations sont effectuées sur le mauvais ensemble d'indices.
- **Erreur de fonction objectif** : le sens de l'optimisation ou certains termes de coût sont mal formulés.

Ces erreurs sont particulièrement critiques en optimisation combinatoire, car une petite erreur de formulation peut rendre toute la solution invalide.

2.6.2 Stratégies de prompting efficaces

Pour réduire ces erreurs, plusieurs stratégies sont recommandées [24] :

- **Décomposer la tâche** : demander d'abord les variables, puis les contraintes, puis la fonction objectif.
- **Spécifier les domaines des variables** : imposer explicitement les domaines binaires, entiers ou continus.
- **Demander une vérification** : inviter le modèle à vérifier si chaque contrainte du problème est bien prise en compte.
- **Imposer un format de sortie** : structurer la réponse pour faciliter le contrôle et le recalcul.
- **Fournir des consignes négatives** : préciser les erreurs à éviter, comme la reproduction de solutions connues ou l'oubli des contraintes.

Ces recommandations justifient plusieurs choix méthodologiques adoptés dans ce travail, notamment l'utilisation d'un rôle d'expert, la décomposition des tâches et l'obligation de présenter les résultats dans des tableaux clairs.

2.6.3 Évaluation des réponses en optimisation

Vu et al. [24] proposent d'évaluer les réponses à plusieurs niveaux : correction syntaxique, correction sémantique et qualité de la solution obtenue après résolution. Cette logique est adaptée dans ce mémoire à travers des critères tels que la faisabilité, la qualité de la solution, le GAP, la fiabilité et le temps de réponse.

L'évaluation ne peut donc pas se limiter à l'apparence de la réponse. Une formulation claire ou une explication convaincante ne garantit pas que la solution soit faisable ou optimale. C'est pourquoi les réponses des LLM doivent être vérifiées à l'aide de scripts ou de solveurs.

2.7 Métriques d'évaluation des prompts

2.7.1 Métriques basées sur la sortie

La qualité d'un prompt est souvent évaluée indirectement par la qualité des réponses qu'il produit [22, 7]. Les métriques dépendent de la tâche considérée. Pour la génération de texte, on peut utiliser des scores comme BLEU ou ROUGE. Pour la génération de code, on peut mesurer le taux de compilation ou le passage de tests unitaires.

Dans ce mémoire, les métriques sont adaptées à l'optimisation combinatoire. Elles portent notamment sur :

- la faisabilité de la solution ;
- la qualité de la solution par rapport à une référence ;
- le GAP ;
- la fiabilité entre ce que le modèle annonce et ce qui est vérifié ;
- la stabilité des réponses ;
- le temps de réponse ;
- le nombre de requêtes nécessaires pour obtenir une solution satisfaisante.

2.7.2 Métriques intrinsèques au prompt

Certains travaux proposent aussi d'évaluer le prompt lui-même avant d'observer la réponse du modèle [21]. Les critères les plus fréquents sont :

- **Clarté** : l'instruction principale est-elle compréhensible et non ambiguë ?
- **Complétude** : toutes les informations nécessaires sont-elles présentes ?
- **Concision** : le prompt évite-t-il les détails inutiles ?
- **Cohérence interne** : les différentes consignes sont-elles compatibles ?

Ces critères sont importants pour construire des prompts expérimentaux reproductibles, surtout lorsque plusieurs modèles doivent être comparés sur les mêmes tâches.

2.7.3 *Prompt Efficiency*

Elnashar et al. [7] introduisent la notion de *prompt efficiency*, qui mesure le rapport entre la qualité de la sortie obtenue et la longueur du prompt utilisé :

$$\eta_{\text{prompt}} = \frac{\text{Score de qualité de la sortie}}{\text{Nombre de tokens du prompt}}$$

Cette notion est pertinente lorsque les appels aux modèles sont coûteux ou limités. Dans le cadre de ce mémoire, elle rejoint l'idée d'efficacité en nombre de requêtes : un modèle qui nécessite plusieurs relances pour atteindre une solution faisable est moins efficace qu'un modèle capable de produire une solution correcte dès les premières interactions.

2.8 Limites du *Prompt Engineering* et défis ouverts

2.8.1 Fragilité des prompts

Un prompt efficace pour un modèle donné peut ne pas fonctionner aussi bien avec un autre modèle, ou même avec une version différente du même modèle [22]. Cette fragilité complique la généralisation des conclusions et impose de documenter précisément les prompts utilisés.

2.8.2 Manque de robustesse

Les réponses des LLM peuvent varier selon la formulation exacte du prompt, les paramètres de génération ou le contexte disponible. Cette variabilité est problématique pour les tâches techniques, où la reproductibilité est une exigence importante.

2.8.3 Absence de garantie formelle

Même un prompt très bien conçu ne garantit pas la faisabilité ou l'optimalité d'une solution. Le modèle peut suivre la structure demandée tout en produisant une solution erronée. En optimisation combinatoire, cette limite est centrale : la validité ne peut pas être déduite de la qualité rédactionnelle de la réponse.

2.8.4 Scalabilité des instances

Les prompts qui fonctionnent sur de petites instances peuvent devenir insuffisants lorsque la taille du problème augmente. Les données peuvent dépasser la fenêtre de contexte, les calculs deviennent plus difficiles, et les risques d'erreur augmentent. Cette limite justifie l'utilisation de benchmarks de tailles variées dans le protocole expérimental.

2.8.5 Nécessité d'une vérification externe

La principale conséquence de ces limites est la nécessité d'une vérification indépendante. Dans ce mémoire, les réponses des LLM sont évaluées à travers des critères objectifs : respect des contraintes, recalcul de la fonction objectif, comparaison avec une solution de référence et analyse du GAP. Le *prompt engineering* permet d'améliorer la qualité des réponses, mais il ne remplace pas la validation formelle.

2.9 Conclusion

Ce chapitre a présenté un état de l'art du *prompt engineering*, en mettant l'accent sur les techniques utiles pour l'utilisation des LLM dans des tâches techniques et vérifiables. Après avoir défini le concept de prompt et expliqué la sensibilité des LLM à leur formulation, nous avons présenté les composantes d'un prompt, les principales familles de techniques, les patterns de conception et les apports spécifiques à l'optimisation mathématique.

Plusieurs enseignements se dégagent de cette revue. Premièrement, le *prompt engineering* ne consiste pas simplement à formuler une question : il s'agit d'une démarche structurée qui combine rôle, contexte, instruction, contraintes et format de sortie. Deuxièmement, les techniques les plus pertinentes pour l'optimisation combinatoire sont celles qui favorisent la décomposition de la tâche, la précision des contraintes, la structuration de la réponse et la vérification. Troisièmement, même un prompt bien conçu ne garantit pas la validité d'une solution : les réponses des LLM doivent être contrôlées par des critères objectifs.

Ces éléments serviront de base au chapitre suivant, qui présente le framework de *prompt engineering* adopté dans ce mémoire pour construire des prompts adaptés aux problèmes étudiés : le p-median, le *Set Covering Problem* et le *Uncapacitated Lot Sizing Problem*.

Chapitre 3

Framework de *Prompt Engineering* pour l'optimisation combinatoire

3.1 Introduction

La conception de prompts efficaces pour des tâches d'optimisation combinatoire ne résulte pas d'une démarche intuitive. Elle est le fruit d'un processus itératif d'expérimentation, d'observation et d'affinement progressif, guidé par des hypothèses sur les capacités et les limites des LLM face à des contraintes mathématiques strictes. Ce chapitre présente le framework de *prompt engineering* développé dans le cadre de ce mémoire pour soumettre aux cinq LLM étudiés des problèmes d'optimisation combinatoire de manière rigoureuse, reproductible et comparable.

Un framework de *prompt engineering* est un ensemble structuré de principes, de règles et de patrons de conception qui guident la formulation des prompts soumis à un LLM pour une catégorie de tâches donnée. Il ne s'agit pas d'un simple ensemble de consignes ad hoc, mais d'une architecture cohérente dont chaque composante est justifiée par des observations empiriques et ancrée dans la littérature scientifique.

Ce chapitre s'organise en quatre parties. La première présente les observations préliminaires qui ont motivé et orienté les choix de conception. La deuxième formalise les sept principes fondateurs du framework. La troisième détaille les trois prompts conçus pour les problèmes du p-médian, du *Set Covering Problem* et du *Uncapacitated Lot Sizing Problem*, avec une analyse composante par composante. La quatrième compare les trois prompts et discute le positionnement du framework proposé par rapport aux travaux existants.

3.2 Observations préliminaires ayant guidé la conception

Avant de concevoir les prompts définitifs, une phase exploratoire a été conduite. Cette phase visait à identifier les facteurs ayant le plus d'influence sur la qualité des réponses dans le contexte spécifique de l'optimisation combinatoire. Trois observations majeures ont émergé, chacune ayant conduit à une décision de conception précise.

3.2.1 Observation 1 : la langue du prompt influence significativement la qualité des réponses

La première série d'expériences a comparé des prompts formulés en français et des prompts formulés en anglais, pour une même instance et un même problème. Les résultats ont montré de manière cohérente que les prompts en anglais produisent des réponses de meilleure qualité selon plusieurs dimensions :

- **Précision de la formulation mathématique** : les modèles formulent plus correctement les variables de décision, les contraintes et la fonction objectif en réponse à des prompts en anglais ;
- **Complétude des réponses** : les réponses en anglais sont plus complètes et respectent mieux la structure demandée ;
- **Cohérence terminologique** : la terminologie de la recherche opérationnelle, comme OR-Library, MILP, Branch-and-Bound, *setup cost* ou *holding cost*, étant majoritairement anglophone dans la littérature scientifique, les modèles y accèdent plus naturellement via des prompts en anglais.

Cette observation s'explique par la distribution des données d'entraînement des LLM : la littérature scientifique en optimisation combinatoire est rédigée majoritairement en anglais, et les modèles ont donc développé des associations sémantiques plus riches et plus précises dans cette langue pour les concepts de recherche opérationnelle.

Décision de conception : tous les prompts définitifs sont rédigés en anglais, indépendamment de la langue de l'utilisateur ou de l'interface utilisée.

3.2.2 Observation 2 : la spécification explicite d'un rôle d'expert améliore la rigueur mathématique

La deuxième série d'expériences a comparé des prompts avec et sans spécification de rôle. Les prompts sans rôle produisaient des réponses parfois correctes sur le fond, mais formulées avec un niveau de généralité insuffisant pour une évaluation rigoureuse : les variables n'étaient pas toujours définies formellement, les contraintes étaient parfois décrites en langage naturel plutôt que mathématiquement, et la méthode de résolution était rarement justifiée.

En revanche, l'ajout de l'instruction suivante :

> *"You are an expert in combinatorial optimization and mathematical programming."*

a produit des réponses plus structurées, avec une notation mathématique plus rigoureuse, une distinction plus claire entre variables de décision, contraintes et objectif, ainsi qu'une argumentation plus détaillée sur le choix de la méthode de résolution.

Ce résultat est cohérent avec les observations de White et al. [26] sur l'effet du *role prompting* : l'assignation d'une expertise active des patterns de génération cohérents

avec le domaine spécifié. Vu et al. [24] confirment également l'importance de la contextualisation domaine pour obtenir des formulations mathématiques correctes en optimisation.

Décision de conception : chaque prompt commence par une spécification explicite et non ambiguë du rôle d'expert en optimisation combinatoire et en programmation mathématique.

3.2.3 Observation 3 : l'interdiction des solutions optimales connues est indispensable pour un raisonnement authentique

La troisième observation est critique pour la validité de l'évaluation. Lors des premières expériences, certains LLM ont produit des solutions correspondant exactement aux optima connus pour des instances de benchmark, sans présenter de véritable raisonnement de résolution. Dans ce cas, le modèle risque de reproduire un résultat mémorisé lors de son pré-entraînement, plutôt que de construire une solution à partir des données de l'instance.

Ce comportement est analogue au phénomène de *data leakage* décrit par Géron [11] en machine learning : une information qui devrait être obtenue par un processus de calcul est disponible directement dans les données utilisées par le modèle, ce qui contamine l'évaluation et rend les résultats non informatifs sur les capacités réelles de raisonnement du LLM.

Pour limiter ce biais, une contrainte explicite a été intégrée à chaque prompt, combinant une formulation négative et une formulation positive :

> *"You are NOT allowed to directly use, recall, or reproduce known optimal solutions or published results associated with this benchmark. The solution must be obtained by explicitly performing the necessary modeling, reasoning, and calculations based solely on the provided data."*

Cette double formulation réduit les ambiguïtés sur le comportement attendu et force le modèle à produire un raisonnement explicite, rendant l'évaluation plus authentique et plus informative.

Décision de conception : chaque prompt inclut une interdiction explicite de reproduire les solutions optimales connues, accompagnée d'une exigence de raisonnement et de calcul explicite basé uniquement sur les données fournies.

3.3 Les sept principes fondateurs du framework

3.3.1 Vue d'ensemble

Le framework développé dans ce mémoire s'appuie sur sept principes fondateurs, dérivés à la fois des observations préliminaires et des meilleures pratiques identifiées dans la littérature [26, 24, 25, 7, 23]. Ces principes s'appliquent uniformément aux trois problèmes étudiés, afin de garantir la cohérence et la comparabilité de l'évaluation.

La Figure 3.1 illustre l'architecture générale du framework et les relations entre observations, principes et prompts.

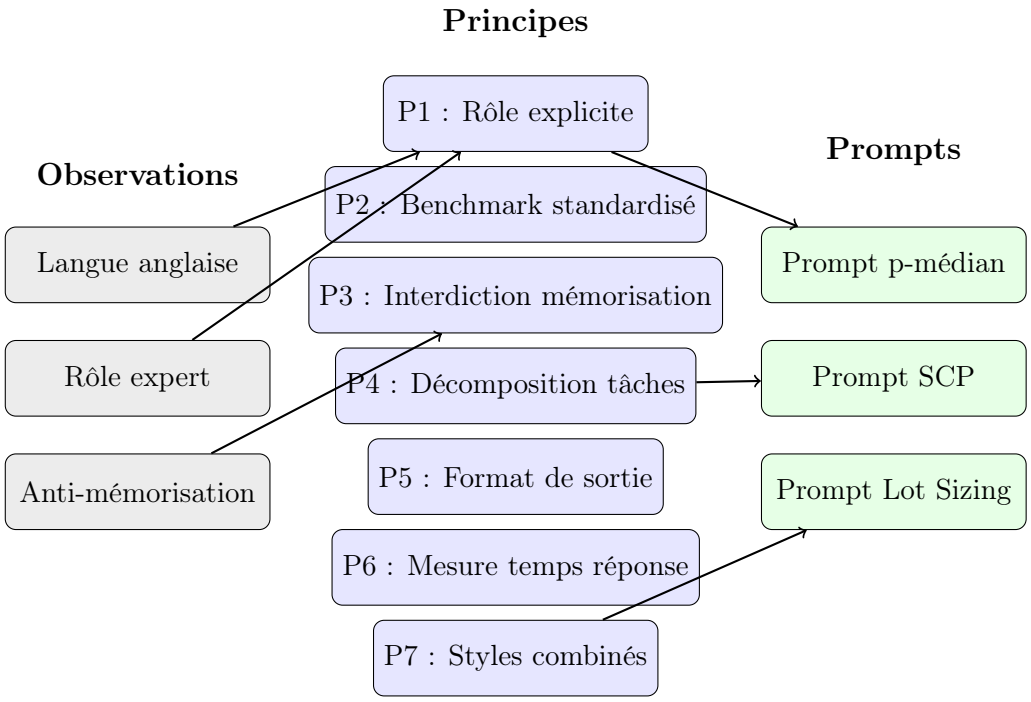


Figure 3.1: Architecture du framework de *prompt engineering* : des observations aux prompts.

3.3.2 Principe 1 : rôle d'expert explicite

Chaque prompt débute par l'assignation d'un rôle d'expert précis et non ambigu, formulé en anglais :

> *"You are an expert in combinatorial optimization and mathematical programming."*

Ce rôle oriente le niveau de technicité de la réponse, favorise l'utilisation d'une terminologie adaptée et réduit les réponses trop générales. Ce principe est directement motivé par l'Observation 2 et s'appuie sur le pattern *Persona* du catalogue de White et al. [26].

3.3.3 Principe 2 : benchmark standardisé et format d'instance explicite

Chaque prompt spécifie explicitement l'instance de benchmark utilisée ainsi que son format de fichier attendu. Cette spécification remplit deux fonctions : elle contextualise la tâche de manière non ambiguë et teste la capacité du modèle à interpréter correctement des données structurées selon un format standardisé de la communauté en recherche opérationnelle.

Ce principe est particulièrement important dans ce mémoire, car les problèmes étudiés reposent sur des instances de benchmark connues, notamment issues de l'OR-Library pour le p-médian et le *Set Covering Problem*.

3.3.4 Principe 3 : interdiction des solutions mémorisées

Chaque prompt inclut une contrainte explicite interdisant la reproduction de solutions publiées, formulée à la fois négativement, par ce qui est interdit, et positivement, par ce qui est exigé. Ce principe est motivé par l'Observation 3 et correspond au *negative prompting* de White et al. [26].

L'objectif est de garantir que l'évaluation porte sur les capacités de formulation, de raisonnement et de calcul du LLM, et non sur une éventuelle mémorisation de résultats connus.

3.3.5 Principe 4 : décomposition structurée de la tâche

Chaque prompt décompose la tâche de résolution en sous-tâches numérotées et séquentielles, couvrant l'ensemble du processus de résolution : analyse des données, formulation mathématique, calcul de la solution, description de la méthode, analyse des performances, justification et discussion.

Cette décomposition correspond au pattern *Cognitive Verifier* de White et al. [26] et aux recommandations de Vu et al. [24] pour réduire les erreurs d'oubli de contraintes. Elle active également un raisonnement de type *Chain-of-Thought* implicite [25].

3.3.6 Principe 5 : spécification explicite du format de sortie

Chaque prompt spécifie le format attendu pour les éléments clés de la réponse : tableau d'affectation pour le p-médian, tableau de couverture pour le SCP, tableau de production pour le Lot Sizing.

Cette spécification facilite la vérification de la faisabilité et la comparaison entre modèles. Elle s'appuie sur les résultats d'Elnashar et al. [7], qui montrent que les prompts avec schéma de sortie explicite produisent une meilleure conformité au format attendu.

3.3.7 Principe 6 : mesure explicite du temps de réponse

Un élément distinctif du framework est la demande explicite de mesure et de report du temps de réponse, avec une distinction entre le temps de génération de la réponse par le LLM et le temps de calcul algorithmique.

Cette distinction permet d'évaluer l'efficacité des modèles en complément de la qualité de leurs solutions. Elle est importante car un modèle peut produire rapidement une réponse, mais cette réponse peut être infaisable ou incorrecte. À l'inverse, un modèle plus lent peut fournir une solution plus structurée ou plus fiable.

3.3.8 Principe 7 : combinaison de styles de prompting

Le framework ne repose pas sur un style de prompting unique. Il combine plusieurs techniques complémentaires issues de la littérature [22, 25] :

- ***Role prompting*** : activation de l'expertise domaine via la spécification du rôle ;
- ***Instruction prompting*** : structuration en sous-tâches numérotées et séquentielles ;
- ***Format prompting*** : spécification des formats de sortie attendus ;
- ***Negative prompting*** : interdiction explicite des comportements indésirables ;
- ***Zero-shot Chain-of-Thought*** implicite : activation du raisonnement étape par étape via la décomposition en tâches séquentielles.

Cette combinaison est motivée par l'observation que chaque technique adresse une faiblesse spécifique des LLM en contexte d'optimisation : le *role prompting* améliore la rigueur mathématique, le *negative prompting* réduit le risque de mémorisation, le *format prompting* facilite la vérification, et la décomposition en tâches réduit les oublis de contraintes.

3.4 Conception et analyse des prompts

3.4.1 Architecture commune aux trois prompts

Les trois prompts partagent une architecture commune en huit sections thématiques, dont la séquence est identique pour garantir la comparabilité :

1. **En-tête** : spécification du rôle et instruction principale ;
2. **Contraintes importantes** : interdiction de mémorisation et exigence de raisonnement explicite ;
3. **Contexte et données** : identification de l'instance et format du fichier ;
4. **Tâche 1 : analyse des données** : parsing, rapport des paramètres, description de la structure ;
5. **Tâche 2 : formulation mathématique** : variables, objectif, contraintes ;
6. **Tâche 3 : solution** : calcul et présentation tabulaire ;
7. **Tâches 4 à 6** : méthode, performance et justification ;
8. **Tâche 7 : mesure du temps de réponse** : distinction entre temps de génération et temps algorithmique.

Cette architecture commune assure que chaque LLM est soumis aux mêmes exigences structurelles, quelle que soit l'instance traitée.

3.4.2 Prompt pour le problème du p-médian

En-tête et rôle

Le prompt p-médian s'ouvre par :

> *"You are an expert in combinatorial optimization and mathematical programming. Solve the pmed1 instance from the OR-Library benchmark and explain your reasoning step by step."*

L'instruction *"step by step"* active implicitement un raisonnement structuré, en incitant le modèle à expliciter chaque étape de sa démarche plutôt que de produire directement une réponse finale.

Section anti-mémorisation

Le prompt inclut ensuite une contrainte anti-mémorisation :

> *"You are NOT allowed to directly use, recall, or reproduce known optimal solutions or published results associated with this benchmark. The solution must be obtained by explicitly performing the necessary modeling, reasoning, and calculations based solely on the provided data."*

La formulation négative est suivie d'une formulation positive, créant une double contrainte qui clarifie le comportement attendu.

Spécification du format OR-Library

Le prompt spécifie le format du fichier pmed1 selon la convention OR-Library :

- première ligne : `n m p` ;
- lignes suivantes : données de distances selon le format de l'instance.

Une clause de repli est prévue si le fichier n'est pas fourni intégralement :

> *"If the exact content of the file is not provided, clearly explain how you would read and parse the pmed1 file according to the OR-Library format."*

Cette clause permet de tester la capacité du modèle à décrire une procédure de lecture et de parsing de l'instance.

Décomposition en tâches

Le prompt décompose la résolution en plusieurs tâches :

1. **Data understanding and parsing** : description du parsing, rapport des valeurs de n, m et p, et description de la structure des distances ;

2. **Mathematical formulation** : définition des variables, fonction objectif, contraintes et domaines binaires ;
3. **Solution** : calcul de la solution, sites sélectionnés, affectations et distance totale ;
4. **Solution method** : méthode utilisée, processus de résolution et critère d'arrêt ;
5. **Performance information** : temps de calcul ou nombre d'itérations ;
6. **Justification and discussion** : justification de la méthode et discussion des limites ;
7. **Response time measurement** : temps de génération de la réponse, distingué du temps de calcul ;
8. **Presentation** : tableau final d'affectation et résumé de la distance totale.

Spécificités du prompt p-médian

Ce prompt présente deux spécificités importantes :

- **La distinction entre temps de réponse et temps de calcul** : elle permet d'évaluer l'efficacité de génération du LLM séparément de la qualité algorithmique de la solution ;
- **L'exigence de description du critère d'arrêt** : elle oblige le modèle à expliciter la logique de résolution utilisée, qu'elle soit exacte, heuristique ou métaheuristique.

3.4.3 Prompt pour le *Set Covering Problem*

Adaptations liées à la structure du SCP

Le *Set Covering Problem* présente des particularités structurelles qui nécessitent des adaptations spécifiques. Le format OR-Library SCP utilise une représentation creuse de la matrice de couverture A :

- première ligne : `m n`, où m est le nombre de lignes à couvrir et n le nombre de colonnes ou ensembles ;
- deuxième ligne : vecteur des coûts $c_1, c_2, \ldots, c_n$;
- lignes suivantes : pour chaque ligne, le nombre de colonnes qui la couvrent, suivi des indices de ces colonnes.

Cette représentation compacte nécessite une reconstruction de la matrice binaire de couverture. Le prompt demande donc explicitement au modèle d'expliquer cette reconstruction :

> *"Explain how the binary coverage matrix A is reconstructed from the file."*

Cette demande permet d'évaluer la capacité du modèle à interpréter des données structurées et à transformer une représentation creuse en formulation mathématique exploitable.

Spécificités de la tâche solution

Pour le SCP, le tableau de solution demandé adopte le format suivant :

```
row | covered by selected sets
```

Ce format permet de vérifier directement la contrainte de couverture complète : chaque ligne doit être couverte par au moins un ensemble sélectionné. Cette structure facilite donc l'analyse de faisabilité de la solution proposée.

Spécificité du prompt SCP

La spécificité principale du prompt SCP est la demande de reconstruction explicite de la matrice A depuis la représentation creuse OR-Library. Cette étape nécessite un raisonnement algorithmique plus précis qu'une simple description générale du problème.

3.4.4 Prompt pour le problème *Uncapacitated Lot Sizing*

Format d'instance spécifique

Le problème *Uncapacitated Lot Sizing* utilise un format d'instance structuré en cinq lignes :

- **Ligne 1** : nombre de périodes T ;
- **Ligne 2** : vecteur de demande $(d_1, d_2, \ldots, d_T)$;
- **Ligne 3** : vecteur des coûts de stockage $(h_1, h_2, \ldots, h_T)$;
- **Ligne 4** : vecteur des coûts de setup $(f_1, f_2, \ldots, f_T)$;
- **Ligne 5** : coût unitaire de production.

Ce format compact teste la capacité du modèle à interpréter correctement chaque paramètre dans son contexte de planification de production.

Triple vérification de faisabilité

Le prompt Lot Sizing inclut une section de vérification de faisabilité plus détaillée que les deux autres prompts, en raison de la nature multi-périodes du problème. Le modèle est invité à vérifier trois éléments :

> *"Verify that: all demands are satisfied, inventory flow is consistent across periods, setup decisions correctly correspond to production."*

Ces vérifications correspondent respectivement à la satisfaction des demandes, à la cohérence du bilan de stock et à la relation entre production et décision de setup.

Format de sortie orienté solveur

Le prompt Lot Sizing demande également une sortie au format solveur :

```
Optimal solution found
Objective value:
Enumerated nodes:
Total iterations:
Time (CPU seconds):
Time (Wallclock seconds):
Statut :  Optimal
Coût total =
```

Ce format impose au modèle de structurer sa réponse de manière standardisée et de distinguer clairement les informations de performance. Il constitue une application des recommandations d'Elnashar et al. [7] sur l'intérêt des schémas de sortie explicites.

Tableau de production structuré

Le prompt Lot Sizing impose un tableau de production précis :

```
| Période | Production | Stock | Setup |
```

Les règles associées sont les suivantes :

- chaque ligne doit correspondre à une période de planification ;
- l'évolution des stocks doit être cohérente entre les périodes ;
- le tableau doit contenir exactement T lignes.

La contrainte sur le nombre exact de lignes constitue une vérification implicite de la compréhension de l'instance par le modèle.

3.5 Comparaison des trois prompts et cohérence du framework

3.5.1 Tableau comparatif des composantes

Table 3.1: Comparaison des composantes des trois prompts conçus

Composante	**Prompt p-médian**	**Prompt SCP**	**Prompt Lot Sizing**
Rôle expert explicite	✓	✓	✓
Interdiction mémorisation	✓	✓	✓
Format OR-Library	✓	✓	× (format 5 lignes)
Formulation mathématique demandée	✓	✓	✓
Tableau de solution structuré	Affectation	Couverture	Production
Triple vérification faisabilité	×	×	✓
Format solveur (*solver-style*)	×	×	✓
Reconstruction matrice creuse	×	✓	×
Mesure temps de réponse	✓	✓	✓
Nombre de tâches	8	8	8

3.5.2 Cohérence et invariants du framework

Le tableau 3.1 met en évidence deux catégories de composantes.

Composantes invariantes. Cinq composantes sont présentes dans les trois prompts : le rôle expert, l'interdiction de mémorisation, la formulation mathématique, la mesure du temps de réponse et la décomposition en tâches. Ces invariants constituent le socle du framework et garantissent la comparabilité des évaluations entre problèmes.

Composantes adaptées. Certaines composantes varient selon la structure du problème : le format d'instance, le type de tableau de solution, la triple vérification de faisabilité pour le Lot Sizing, le format solveur pour le Lot Sizing et la reconstruction de matrice creuse pour le SCP. Ces adaptations montrent que le framework est à la fois stable dans ses principes et flexible dans son application.

3.5.3 Positionnement du framework proposé par rapport aux travaux existants

Le framework proposé s'inscrit dans la continuité des travaux existants sur le *prompt engineering*, tout en introduisant des adaptations spécifiques au contexte de l'optimisation combinatoire. Contrairement aux approches générales de la littérature, qui se concentrent souvent sur la qualité textuelle des réponses, la génération de code ou la structuration des sorties, ce framework met l'accent sur des critères directement vérifiables : faisabilité, respect des contraintes, valeur de la fonction objectif, GAP, fiabilité et temps de réponse.

Par rapport aux travaux de Vu et al. [24], centrés sur la formulation de problèmes d'optimisation à l'aide de LLM, le framework proposé va plus loin en intégrant non seulement la formulation mathématique, mais aussi la résolution, la présentation de la solution, la justification de la méthode utilisée et la vérification des résultats produits. Il ajoute également une contrainte anti-mémorisation, afin d'éviter que les modèles ne reproduisent directement des solutions optimales connues des benchmarks.

Par rapport au catalogue de patterns de White et al. [26], le framework ne se limite pas à l'application isolée d'un seul pattern. Il combine plusieurs techniques complémentaires, notamment le *Persona pattern*, le *Template pattern*, le *Cognitive Verifier pattern*, le *Fact Check List pattern* et le *negative prompting*. Cette combinaison permet de structurer la réponse du LLM, de limiter certains comportements indésirables et de faciliter la vérification des solutions proposées.

Par rapport aux travaux d'Elnashar et al. [7], qui étudient l'effet des formats de prompts sur la génération de données structurées, le framework proposé adapte cette logique à des problèmes d'optimisation. Dans ce contexte, la conformité au format attendu ne suffit pas : une solution peut être bien présentée tout en étant infaisable ou incorrecte. C'est pourquoi le framework impose également des éléments de vérification liés aux contraintes, aux affectations, aux coûts et à la cohérence des décisions.

Ainsi, la contribution principale de ce framework réside dans son adaptation au contexte particulier de l'optimisation combinatoire. Il ne vise pas seulement à obtenir des réponses claires ou bien structurées, mais à produire des réponses évaluables, comparables et vérifiables. Cette orientation distingue le framework proposé des approches plus générales du *prompt engineering* et justifie son utilisation dans le protocole expérimental présenté au chapitre suivant.

3.6 Conclusion

Ce chapitre a présenté le framework de *prompt engineering* développé pour l'évaluation comparative des LLM sur des problèmes d'optimisation combinatoire. Ce framework est le fruit d'une démarche empirique rigoureuse, guidée par trois observations préliminaires fondamentales : la supériorité des prompts en anglais, l'apport du *role prompting*, et la nécessité d'interdire la reproduction de solutions mémorisées. Il s'appuie également sur les meilleures pratiques de la littérature [26, 24, 25, 7, 23, 22].

Les sept principes fondateurs du framework constituent un cadre cohérent, reproductible et justifié pour la conception de prompts en contexte d'optimisation sous contraintes. Les trois prompts développés pour le p-médian, le SCP et le Lot Sizing appliquent ces principes de manière uniforme sur les invariants, tout en intégrant des adaptations spécifiques à la structure de chaque problème.

Le tableau comparatif présenté dans la section précédente synthétise les choix de conception et met en évidence la cohérence interne du framework : certaines composantes garantissent la comparabilité entre problèmes, tandis que d'autres reflètent les spécificités structurelles des problèmes traités. Ces prompts seront utilisés dans le chapitre expérimental pour soumettre les trois problèmes aux cinq LLM étudiés.

Chapitre 4

Évaluation des LLM comme solveurs autonomes

4.1 Introduction

Ce chapitre présente la première phase de notre travail expérimental. Cette phase vise à évaluer la capacité des grands modèles de langage (*Large Language Models*, LLM) à résoudre, de manière autonome, des problèmes classiques d'optimisation combinatoire appliqués à la logistique et à la gestion de production.

Cette étude s'inscrit dans un contexte où les LLM deviennent de plus en plus accessibles à des utilisateurs non experts. Dans le domaine industriel, de nombreux praticiens, tels que les responsables logistiques, les gestionnaires de production ou les planificateurs, sont confrontés à des problèmes d'optimisation complexes sans nécessairement disposer de compétences avancées en recherche opérationnelle ou en programmation.

Face à l'essor d'outils conversationnels grand public comme ChatGPT, Gemini, DeepSeek, Grok ou Claude, une question importante se pose : **un utilisateur non expert peut-il soumettre directement un problème d'optimisation à un LLM et obtenir une solution fiable, faisable et de bonne qualité ?**

C'est précisément à cette question que cette première phase cherche à répondre. L'objectif est d'adopter volontairement le point de vue d'un utilisateur non expert. Les modèles reçoivent uniquement la description du problème et les données associées. Aucun solveur externe, aucun outil de vérification, aucune formulation mathématique détaillée et aucune méthode de résolution guidée ne sont fournis au LLM au moment de la résolution.

L'intérêt de cette approche est de tester les capacités propres des LLM lorsqu'ils sont utilisés comme des solveurs autonomes. Les réponses produites sont ensuite évaluées à l'aide d'indicateurs permettant de mesurer la qualité des solutions, leur faisabilité, leur fiabilité, leur stabilité et leur capacité à s'améliorer au fil des interactions.

4.2 Problématique et objectifs

4.2.1 Contexte et motivation

Les problèmes d'optimisation combinatoire sont présents dans de nombreuses activités industrielles et logistiques : localisation d'entrepôts, planification de production, gestion des stocks, affectation de ressources ou encore couverture de zones géographiques. Ces problèmes sont généralement résolus par des experts en recherche opérationnelle à l'aide de solveurs spécialisés, tels que CPLEX, Gurobi ou OR-Tools, ou à l'aide de méthodes heuristiques adaptées au problème étudié.

Cependant, ces outils restent souvent difficiles à utiliser pour des utilisateurs non spécialistes. Leur mise en œuvre demande généralement une formulation mathématique rigoureuse, une bonne connaissance des solveurs et des compétences en programmation. En pratique, cela limite leur utilisation directe par des acteurs opérationnels qui ont pourtant besoin de prendre des décisions efficaces.

Les LLM changent en partie cette situation. Leur capacité à comprendre le langage naturel, à expliquer des raisonnements et à générer des solutions ou du code donne l'impression qu'ils peuvent aider à résoudre des problèmes complexes. Il devient donc nécessaire d'évaluer rigoureusement ce qu'ils peuvent réellement faire dans un contexte d'optimisation, au-delà de la qualité apparente de leurs réponses.

4.2.2 Question de recherche

La question centrale de cette phase est la suivante :

> *Dans quelle mesure un LLM grand public, utilisé de manière autonome par un utilisateur non expert, est-il capable de produire des solutions faisables et de qualité satisfaisante pour des problèmes d'optimisation combinatoire liés à la logistique et à la gestion de production ?*

Cette question implique de ne pas seulement vérifier si le LLM donne une réponse, mais aussi de contrôler si cette réponse est correcte, si elle respecte les contraintes du problème et si les affirmations du modèle sont cohérentes avec les résultats réellement vérifiés.

4.2.3 Objectifs spécifiques

Cette phase poursuit trois objectifs principaux :

- **Évaluer les capacités intrinsèques des LLM** à résoudre des problèmes d'optimisation combinatoire sans outil externe ni aide méthodologique détaillée.
- **Comparer les performances de cinq LLM grand public** sur plusieurs familles de problèmes représentatives de la logistique et de la gestion de production.
- **Identifier les limites et les comportements récurrents** des LLM face à des instances de tailles différentes, notamment en termes de faisabilité, de fiabilité, d'hallucinations et d'erreurs de calcul.

4.3 Protocole expérimental

4.3.1 Vue d'ensemble du protocole

Le protocole expérimental est conçu pour simuler le comportement d'un utilisateur non expert soumettant un problème d'optimisation à un LLM via une interface conversationnelle standard. Le modèle reçoit une description du problème ainsi que les données nécessaires, puis doit produire une solution en autonomie.

Chaque expérience suit les étapes suivantes :

1. Sélection d'une instance de benchmark.
2. Construction du prompt selon le cadre de formulation présenté dans le Chapitre 3.
3. Soumission du prompt au LLM dans deux configurations : avec rôle et sans rôle.
4. Collecte de la réponse produite par le modèle.
5. Vérification externe de la solution proposée.
6. Évaluation de la réponse selon les indicateurs définis dans la Section 4.6.

Même si le LLM n'a pas accès à un outil externe au moment de produire sa réponse, les solutions sont ensuite vérifiées à l'aide d'outils de calcul. Cette vérification est indispensable pour mesurer objectivement la qualité des réponses et éviter de se limiter aux affirmations du modèle.

4.3.2 Design expérimental

Le design expérimental est structuré selon quatre dimensions : les problèmes testés, les modèles évalués, les instances utilisées et les configurations de prompt.

Table 4.1: Design expérimental de la Phase 1

Dimension	**Modalités**
Problèmes testés	Uncapacitated P-Median Problem (UPMP) Set Covering Problem (SCP) Uncapacitated Lot Sizing Problem (ULS)
LLM testés	ChatGPT 5.3, Gemini 3, DeepSeek, Grok 4, Claude Sonnet 4.6
Instances par problème	15 instances par problème
Configurations de prompt	Avec rôle / Sans rôle

Le nombre total d'expériences prévu dans cette phase est donc :

$$3 \text{ problèmes} \times 15 \text{ instances} \times 5 \text{ LLM} \times 2 \text{ configurations} = 450 \text{ expériences}$$

Les expérimentations préliminaires ont permis de comparer les deux configurations de prompt. Les résultats détaillés présentés dans les sections suivantes portent principalement sur la configuration **avec rôle**, car elle a donné des réponses globalement plus exploitables que la configuration sans rôle.

4.4 Benchmarks et problèmes étudiés

4.4.1 Uncapacitated P-Median Problem (UPMP)

Description du problème

Le problème de la p-médiane non capacitée (*Uncapacitated P-Median Problem*, UPMP) est un problème classique de localisation. Il consiste à sélectionner p sites parmi un ensemble de sites candidats afin de minimiser la somme des distances entre chaque client et le site sélectionné le plus proche.

Une formulation classique du problème est la suivante :

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$$

$$\text{s.c.} \quad \sum_{j=1}^{n} x_{ij} = 1 \quad \forall i$$

$$x_{ij} \leq y_j \quad \forall i, j$$

$$\sum_{j=1}^{n} y_j = p$$

$$x_{ij}, y_j \in \{0, 1\}$$

où d_{ij} représente le coût ou la distance entre le client i et le site j, y_j vaut 1 si le site j est sélectionné, et x_{ij} vaut 1 si le client i est affecté au site j.

Pertinence logistique

Ce problème représente des décisions stratégiques fréquentes en logistique, comme la localisation d'entrepôts, de dépôts, de centres de service ou d'agences de distribution. Il est donc directement lié à des situations où un décideur doit choisir un nombre limité de sites afin de réduire les coûts de transport ou de service.

Source et instances utilisées

Les instances utilisées proviennent de l'OR-Library. Les valeurs optimales connues sont utilisées comme références pour le calcul du GAP.

Table 4.2: Instances UPMP utilisées

Groupe	**Instances**	**Nœuds (n)**	p	**Nb.**
Petites instances	pmed1, pmed4, pmed7, pmed10, pmed13	100–300	5–30	5
Moyennes instances	pmed15, pmed17, pmed20, pmed23, pmed27	300–600	10–100	5
Grandes instances	pmed30, pmed32, pmed35, pmed38, pmed40	600–900	90–200	5
Total	—	—	—	**15**

4.4.2 Set Covering Problem (SCP)

Description du problème

Le problème de couverture par ensembles (*Set Covering Problem*, SCP) consiste à sélectionner un sous-ensemble de colonnes d'une matrice binaire afin de couvrir toutes les lignes, tout en minimisant le coût total des colonnes sélectionnées.

Une formulation classique du SCP est la suivante :

$$\min \sum_{j=1}^{n} c_j x_j$$

$$\text{s.c.} \quad \sum_{j=1}^{n} a_{ij} x_j \geq 1 \quad \forall i \in \{1, \ldots, m\}$$

$$x_j \in \{0, 1\} \quad \forall j$$

où c_j est le coût de sélection de la colonne j, $a_{ij} = 1$ si la colonne j couvre la ligne i, et $x_j = 1$ si la colonne j est sélectionnée.

Pertinence logistique

Le SCP modélise plusieurs décisions de couverture rencontrées en logistique et en organisation industrielle : affectation de ressources, couverture de zones géographiques, sélection de services, planification de tâches ou encore conception de tournées couvrant un ensemble de demandes.

Source et instances utilisées

Les instances utilisées proviennent de l'OR-Library. Les résultats de référence sont obtenus à l'aide d'un code Python de vérification et de résolution développé pour ce travail. Les instances SCP ne sont pas regroupées selon une taille croissante, car certaines familles possèdent les mêmes dimensions. Elles sont donc classées selon leur famille d'origine dans l'OR-Library.

Table 4.3: Instances SCP utilisées

Groupe	**Instances**	**Lignes** (m)	**Colonnes** (n)	**Nb.**
Famille scp4	scp41, scp43, scp45, scp47, scp49	200	1000	5
Famille scp5	scp51, scp53, scp55, scp57, scp59	200	2000	5
Famille scp6	scp61, scp62, scp63, scp64, scp65	200	1000	5
Total	—	—	—	**15**

Dans le cas du SCP, les groupes sont présentés selon les séries du benchmark OR-Library. Cette distinction est plus adaptée qu'une simple classification par taille, car les séries ne se différencient pas uniquement par le nombre de lignes et de colonnes.

4.4.3 Uncapacitated Lot Sizing Problem (ULS)

Description du problème

Le problème de dimensionnement de lots non capacitaire (*Uncapacitated Lot Sizing Problem*, ULS) est un problème classique de planification de production. Il consiste à déterminer, pour chaque période d'un horizon de planification, les quantités à produire et les niveaux de stock, de manière à satisfaire une demande connue tout en minimisant les coûts de production, de stockage et de lancement.

Une formulation classique du problème est la suivante :

$$\min \sum_{t=1}^{T} (p_t x_t + h_t s_t + f_t y_t)$$

$$\text{s.c.} \quad s_{t-1} + x_t = d_t + s_t \quad \forall t \in \{1, \ldots, T\}$$

$$x_t \leq M y_t \quad \forall t$$

$$x_t \geq 0, \quad s_t \geq 0, \quad y_t \in \{0, 1\} \quad \forall t$$

où x_t représente la quantité produite à la période t, s_t le niveau de stock, d_t la demande, p_t le coût de production, h_t le coût de stockage, f_t le coût de lancement, et y_t une variable binaire indiquant si une production est lancée à la période t.

Pertinence logistique

Ce problème est directement lié à la planification de production. Il permet de modéliser des décisions telles que : quand produire, combien produire, et comment limiter les coûts tout en satisfaisant la demande. Il est donc particulièrement pertinent pour des responsables de production ou des planificateurs industriels.

Source et instances utilisées

Les instances utilisées proviennent d'un dépôt de benchmark dédié au lot sizing. Les solutions de référence sont obtenues à l'aide d'un code Python développé pour ce travail.

Table 4.4: Instances ULS utilisées

Groupe	**Instances**	**Périodes (T)**	**Nb.**
Petites instances	uls_toy, uls_21.1, uls_60.1, uls_60.2, uls_60.5	7–60	5
Moyennes instances	uls_90.1, uls_90.3, uls_90.5, uls_90.7, uls_90.9	90	5
Grandes instances	uls_120.2, uls_120.4, uls_120.6, uls_120.8, uls_120.10	120	5
Total	—	—	**15**

4.5 LLM évalués

Cinq LLM grand public ont été sélectionnés pour cette étude : **ChatGPT 5.3**, **Gemini 3**, **DeepSeek**, **Grok 4** et **Claude Sonnet 4.6**. Le choix de ces modèles repose sur leur accessibilité via des interfaces conversationnelles, leur popularité auprès du grand public et leur utilisation fréquente pour des tâches de raisonnement, de calcul, de génération de code et d'aide à la décision.

Dans cette phase, l'objectif n'est pas d'étudier l'architecture interne de chaque modèle, mais d'évaluer leur comportement du point de vue d'un utilisateur non expert. Les modèles sont donc considérés comme des outils conversationnels disponibles en ligne, utilisés sans API, sans paramétrage avancé et sans intégration à un solveur externe.

Table 4.5: LLM évalués dans la Phase 1

LLM	**Éditeur**	**Rôle dans l'évaluation**
ChatGPT 5.3	OpenAI	Modèle conversationnel généraliste, utilisé pour tester la rapidité, la structuration des réponses et la capacité à proposer des solutions d'optimisation.
Gemini 3	Google DeepMind	Modèle généraliste utilisé pour évaluer le raisonnement, la stabilité des réponses et la capacité à traiter des données structurées.
DeepSeek	DeepSeek AI	Modèle utilisé pour tester les capacités de raisonnement mathématique, de calcul et de résolution structurée.
Grok 4	xAI	Modèle utilisé pour évaluer la capacité à produire des solutions optimales ou proches de l'optimum dans un contexte conversationnel.
Claude Sonnet 4.6	Anthropic	Modèle utilisé pour évaluer la qualité du raisonnement, la faisabilité des solutions et la fiabilité des réponses.

Cette présentation volontairement synthétique permet d'éviter de fonder l'analyse sur des caractéristiques techniques susceptibles d'évoluer. Ce sont les réponses produites dans le cadre expérimental qui servent de base à la comparaison.

4.6 Indicateurs d'évaluation

Afin d'évaluer les performances des LLM de manière rigoureuse, un ensemble de **12 indicateurs** a été utilisé. Ces indicateurs permettent de mesurer non seulement la qualité mathématique des solutions, mais aussi leur faisabilité, la cohérence du raisonnement, la fiabilité des affirmations du modèle, la stabilité du comportement et l'efficacité pratique de la réponse.

4.6.1 Qualité de la solution

Table 4.6: Indicateurs liés à la qualité de la solution

Indicateur	Définition	Mesure
Solution Quality	Valeur de la solution proposée par le LLM après la première réponse.	Valeur de la fonction objectif
Optimality First Iteration	Indique si le LLM atteint l'optimum dès la première réponse.	Oui / Non
GAP (%)	Écart relatif entre la solution proposée et la meilleure solution de référence.	Pourcentage

Le GAP est calculé selon la formule suivante :

$$\text{GAP} = \frac{|z_{LLM} - z^*|}{z^*} \times 100$$

où z_{LLM} représente la valeur de la solution proposée par le LLM, et z^* la valeur optimale ou de référence.

4.6.2 Faisabilité et validité

Table 4.7: Indicateurs liés à la faisabilité et à la validité

Indicateur	Définition	Mesure
Feasibility	Vérifie si la solution proposée respecte toutes les contraintes du problème.	Oui / Non
Capability	Qualifie la capacité du modèle à produire une solution optimale, faisable ou infaisable.	Optimal / Faisable / Infaisable
Objective Recalculated	Valeur réelle de la fonction objectif après vérification externe de la solution.	Valeur numérique
Reliability	Vérifie si ce que le LLM affirme correspond au résultat réellement obtenu après contrôle.	Oui / Non

La distinction entre **Feasibility** et **Reliability** est essentielle. Un modèle peut proposer une solution faisable, mais annoncer une mauvaise valeur objectif. Il peut aussi affirmer qu'une solution est optimale alors qu'elle ne l'est pas. Dans ces cas, le modèle n'est pas considéré comme fiable, même si la réponse semble bien structurée.

4.6.3 Raisonnement et comportement

Table 4.8: Indicateurs liés au raisonnement et au comportement

Indicateur	Définition	Mesure
Reasoning Comprehension	Analyse de la méthode utilisée par le LLM et de la cohérence de son raisonnement.	Évaluation qualitative
Stability	Vérifie si le modèle garde un comportement cohérent lorsque le prompt est répété.	Oui / Non
Improve Learning	Vérifie si le modèle améliore sa solution après une nouvelle interaction ou une correction.	Oui / Non

4.6.4 Efficacité pratique

Table 4.9: Indicateurs liés à l'efficacité pratique

Indicateur	Définition	Mesure
Response Time	Temps nécessaire au modèle pour produire sa réponse.	Secondes
Query Efficiency	Nombre de requêtes nécessaires pour atteindre une solution optimale ou satisfaisante.	Nombre de prompts

4.7 Outils utilisés

Python et Microsoft Excel ont été utilisés uniquement après la génération des réponses par les LLM. Ils ne constituent donc pas une aide fournie aux modèles pendant la résolution, mais servent à vérifier, organiser et analyser les résultats expérimentaux.

Python a été utilisé pour charger les instances de benchmark, vérifier la *Feasibility* des solutions proposées, recalculer la valeur réelle de la fonction objectif, comparer les résultats avec les valeurs de référence et calculer les indicateurs quantitatifs, notamment le *GAP*. Cette étape est indispensable, car une réponse bien formulée par un LLM peut rester mathématiquement incorrecte.

Microsoft Excel a ensuite été utilisé pour structurer les résultats par problème, instance et modèle, enregistrer les valeurs vérifiées, comparer les indicateurs entre les LLM et construire les tableaux de synthèse présentés dans ce chapitre.

4.8 Résultats expérimentaux du problème p-median

Cette section présente les résultats obtenus pour le problème du p-median dans la première phase expérimentale. Les résultats analysés ici correspondent à la configuration

avec rôle, puisque les essais préliminaires ont montré que cette configuration donnait globalement des réponses plus exploitables que la configuration sans rôle.

L'analyse porte sur 15 instances issues de l'OR-Library. Les performances des cinq LLM sont comparées selon plusieurs indicateurs : le GAP moyen, l'optimalité, la faisabilité, la fiabilité, la stabilité, la capacité d'amélioration et le temps de réponse. L'objectif n'est pas seulement de vérifier si les modèles donnent une réponse, mais surtout de déterminer si cette réponse est correcte, faisable et cohérente avec ce que le modèle affirme.

4.8.1 Comparaison des GAP moyens

Dans cette analyse, le GAP moyen est calculé sur l'ensemble des **15 instances**. Lorsqu'une valeur numérique de GAP est disponible, elle est utilisée directement. En revanche, lorsqu'aucune valeur de GAP n'est renseignée, notamment en raison d'une solution infaisable, non vérifiable ou non exploitable, une valeur pénalisante de **100 %** est attribuée.

Table 4.10: Comparaison des GAP moyens pénalisés des LLM sur le problème du p-median

Rang	**LLM**	**Average GAP pénalisé**
1	Claude Sonnet 4.6	6,69 %
2	Gemini 3	8,44 %
3	DeepSeek	9,39 %
4	Grok 4	13,92 %
5	ChatGPT 5.3	14,68 %

D'après le tableau 4.10, **Claude Sonnet 4.6** obtient le meilleur GAP moyen pénalisé sur le problème du p-median, avec **6,69 %**. Ce résultat montre qu'il est le modèle dont les solutions sont globalement les plus proches des solutions de référence, tout en tenant compte des cas où aucune solution exploitable n'est disponible.

Gemini 3 arrive en deuxième position avec un GAP moyen pénalisé de **8,44 %**. Ce résultat indique que ses solutions restent relativement proches de l'optimum en moyenne. Il montre donc une bonne qualité globale sur cet indicateur, même si ses performances doivent être croisées avec les autres critères, notamment la fiabilité.

DeepSeek obtient un GAP moyen pénalisé de **9,39 %**. Il se place donc en troisième position sur cet indicateur. Ce résultat suggère que, malgré ses limites sur d'autres critères, ses solutions restent en moyenne assez proches des solutions de référence lorsque l'on considère le GAP pénalisé.

Grok 4 présente un GAP moyen pénalisé de **13,92 %**. Il reste compétitif, mais ses solutions sont en moyenne plus éloignées de l'optimum que celles de Claude Sonnet 4.6, Gemini 3 et DeepSeek. Ce résultat doit toutefois être mis en relation avec ses bons résultats en fiabilité et en amélioration au fil des interactions.

Enfin, **ChatGPT 5.3** obtient le GAP moyen pénalisé le plus élevé, avec **14,68 %**. Il

peut produire des solutions exploitables, mais leur proximité moyenne avec l'optimum reste plus limitée par rapport aux autres modèles évalués.

Cette analyse montre que, pour le problème du p-median, le GAP moyen pénalisé permet une comparaison plus équilibrée entre les modèles. Il tient compte non seulement de la qualité des solutions numériques obtenues, mais aussi des cas où les solutions ne sont pas exploitables. Ainsi, **Claude Sonnet 4.6** se distingue comme le modèle le plus performant sur cet indicateur, suivi par **Gemini 3** et **DeepSeek**.

4.8.2 Analyse de l'optimalité

L'optimalité est analysée à travers deux indicateurs : **Optimality First Iteration**, qui indique si le modèle atteint l'optimum dès la première réponse, et **Capability**, qui indique si le modèle est finalement classé comme ayant obtenu une solution optimale.

Table 4.11: Nombre de solutions optimales obtenues par les LLM sur le problème du p-median

LLM	**Capability = Optimal**
Claude Sonnet 4.6	7 / 15
Gemini 3	2 / 15
Grok 4	4 / 15
ChatGPT 5.3	2 / 15
DeepSeek	0 / 15

D'après les résultats présentés dans le tableau 4.11, **Claude Sonnet 4.6** est le modèle qui atteint le plus souvent l'optimum sur le problème du p-median, avec **7 solutions optimales sur 15**. Il se distingue donc clairement des autres modèles sur l'indicateur *Capability = Optimal*.

Grok 4 arrive en deuxième position avec **4 solutions optimales sur 15**. Ce résultat montre qu'il possède une certaine capacité à identifier des solutions optimales, même s'il reste nettement moins performant que Claude Sonnet 4.6 sur cet indicateur.

Gemini 3 et **ChatGPT 5.3** obtiennent chacun **2 solutions optimales sur 15**. Leur performance en matière d'optimalité reste donc limitée, malgré leur capacité à produire des réponses structurées et parfois exploitables.

Enfin, **DeepSeek** n'obtient aucune solution optimale parmi les instances testées. Ce résultat confirme ses limites sur le problème du p-median, en particulier lorsqu'il s'agit d'atteindre exactement la solution optimale.

Globalement, **Claude Sonnet 4.6** apparaît comme le modèle le plus performant en termes d'optimalité sur le problème du p-median. **Grok 4** constitue le deuxième modèle le plus solide, tandis que Gemini 3, ChatGPT 5.3 et DeepSeek présentent des performances plus faibles sur cet indicateur.

4.8.3 Analyse de la fiabilité

L'indicateur **Reliability** permet de vérifier si ce que le modèle annonce correspond réellement aux résultats vérifiés. Cet indicateur est important, car un LLM peut affirmer qu'une solution est optimale ou faisable alors que le contrôle externe montre le contraire.

Table 4.12: Comparaison de la fiabilité des LLM sur le problème du p-median

Rang	**LLM**	**Reliability = Yes**	**Percentage**
1	Grok 4	7 / 15	46.67 %
2	Claude Sonnet 4.6	6 / 15	40.00 %
3	ChatGPT 5.3	1 / 15	6.67 %
4	Gemini 3	0 / 15	0.00 %
5	DeepSeek	0 / 15	0.00 %

Selon cet indicateur, **Grok 4** est le modèle le plus fiable, avec **7 cas de Reliability = yes sur 15**. Il est suivi par **Claude Sonnet 4.6**, qui obtient **6 cas sur 15**.

ChatGPT 5.3 n'obtient qu'un seul cas de fiabilité positive. Cela montre que ses réponses peuvent être structurées et rapides, mais que ses affirmations ne correspondent pas toujours aux résultats réellement vérifiés.

Gemini 3 et **DeepSeek** n'obtiennent aucun cas de fiabilité positive. Leurs réponses doivent donc être interprétées avec beaucoup de prudence, même lorsqu'elles semblent cohérentes sur le plan textuel.

Cette analyse montre que la qualité apparente d'une réponse ne suffit pas. Un modèle peut produire une réponse convaincante, mais annoncer une valeur objectif incorrecte, une solution faussement optimale ou une faisabilité non vérifiée.

4.8.4 Amélioration au fil des itérations

L'indicateur **Improve Learning** permet d'observer si le modèle améliore sa solution après une nouvelle interaction, une répétition du prompt ou une correction.

Table 4.13: Amélioration des LLM au fil des itérations sur le problème du p-median

Rang	**LLM**	**Improve Learning = Yes**	**Instances**
1	Grok 4	5 / 15	pmed7, pmed10, pmed13, pmed17, pmed20
2	DeepSeek	2 / 15	pmed1, pmed30
3	ChatGPT 5.3	1 / 15	pmed13
3	Claude Sonnet 4.6	1 / 15	pmed13
5	Gemini 3	0 / 15	Aucune

Grok 4 est le modèle qui s'améliore le plus souvent au fil des itérations, avec **5 cas d'amélioration sur 15**. Cela montre qu'il peut parfois corriger ou ajuster sa réponse lorsqu'il reçoit une interaction supplémentaire.

DeepSeek montre une amélioration dans **2 cas**, mais cette amélioration reste limitée. **ChatGPT 5.3** et **Claude Sonnet 4.6** ne montrent chacun qu'un seul cas d'amélioration, sur l'instance **pmed13**. Pour Claude, ce résultat doit être interprété avec prudence, car il obtient déjà de bons résultats dès les premières réponses.

Gemini 3 ne montre aucun cas d'amélioration dans les résultats observés. Cela signifie que la répétition ou la correction du prompt ne permet pas forcément d'améliorer ses solutions sur le p-median.

4.8.5 Temps de réponse

Le temps de réponse permet d'évaluer la rapidité avec laquelle chaque modèle produit une solution. Cet indicateur est utile dans un contexte pratique, mais il doit toujours être interprété avec la qualité, la faisabilité et la fiabilité de la solution.

Table 4.14: Classement des LLM selon le temps de réponse sur le problème du p-median

Rang	**LLM**	**Average Response Time**	**Minimum Response Time**	**Maximum Response Time**	**Ranking**
1	ChatGPT 5.3	34.93 s	10 s	100 s	Le plus rapide
2	Gemini 3	49.00 s	10 s	145 s	Rapide
3	DeepSeek	91.40 s	7 s	215 s	Moyen
4	Grok 4	265.00 s	3 s	480 s	Lent
5	Claude Sonnet 4.6	319.13 s	19 s	600 s	Le plus lent

ChatGPT 5.3 est le modèle le plus rapide, avec un temps moyen de **34.93 secondes**. Il est suivi par **Gemini 3**, avec **49 secondes** en moyenne.

DeepSeek occupe une position intermédiaire, avec un temps moyen de **91.40 secondes**. **Grok 4** et **Claude Sonnet 4.6** sont nettement plus lents, avec respectivement **265 secondes** et **319.13 secondes** en moyenne.

Cependant, le classement par temps de réponse ne correspond pas au classement par qualité de solution. ChatGPT est le plus rapide, mais il présente un GAP moyen élevé et une faible fiabilité. À l'inverse, Claude est le plus lent, mais il obtient les meilleures performances en qualité et en optimalité.

4.8.6 Comparaison globale des LLM sur le p-median

Le tableau suivant synthétise les principaux résultats obtenus par les cinq LLM sur le problème du p-median.

Table 4.15: Comparaison globale des LLM sur le problème du p-median

LLM	Average GAP pénalisé	Capability = Optimal	Feasibility = Yes	Reliability = Yes	Stability = Yes	Improve Learning	Average Response Time
ChatGPT 5.3	14,68 %	2 / 15	14 / 15	1 / 15	9 / 15	1 / 15	34,93 s
Gemini 3	8,44 %	2 / 15	14 / 15	0 / 15	14 / 15	0 / 15	49,00 s
DeepSeek	9,39 %	0 / 15	14 / 15	0 / 15	9 / 15	2 / 15	91,40 s
Grok 4	13,92 %	4 / 15	13 / 15	7 / 15	10 / 15	5 / 15	265,00 s
Claude Sonnet 4.6	6,69 %	7 / 15	14 / 15	6 / 15	12 / 15	1 / 15	319,13 s

Cette comparaison montre qu'aucun modèle ne domine l'ensemble des critères évalués. Les performances varient selon l'indicateur considéré : qualité de solution, optimalité, faisabilité, fiabilité, stabilité, capacité d'amélioration ou temps de réponse.

Claude Sonnet 4.6 apparaît comme le modèle le plus performant globalement sur le problème du p-median. Il obtient le plus faible GAP moyen pénalisé, avec **6,69 %**, ainsi que le plus grand nombre de solutions optimales selon l'indicateur *Capability = Optimal*, avec **7 solutions optimales sur 15**. Il présente également une bonne stabilité, avec **12 cas stables sur 15**. En revanche, son temps de réponse moyen est le plus élevé, avec **319,13 secondes**.

Gemini 3 obtient le deuxième meilleur GAP moyen pénalisé, avec **8,44 %**. Il se distingue aussi par une très forte stabilité, avec **14 cas stables sur 15**, ainsi qu'une faisabilité élevée, avec **14 solutions faisables sur 15**. Cependant, sa fiabilité est nulle selon l'indicateur *Reliability = Yes*. Ce résultat montre que, malgré une bonne proximité moyenne avec les solutions de référence, ses réponses doivent être vérifiées avec attention.

DeepSeek présente un GAP moyen pénalisé de **9,39 %**, ce qui le place en troisième position sur cet indicateur. Il produit également **14 solutions faisables sur 15**. Toutefois, il n'obtient aucune solution optimale selon *Capability* et aucun cas fiable selon *Reliability*. Ses résultats montrent donc une certaine proximité moyenne avec les solutions de référence, mais une faible capacité à atteindre l'optimum et à fournir des réponses fiables.

Grok 4 se distingue surtout par sa fiabilité et sa capacité d'amélioration. Il obtient le meilleur score de fiabilité, avec **7 cas fiables sur 15**, ainsi que le meilleur résultat en *Improve Learning*, avec **5 améliorations sur 15**. Il atteint également l'optimum dans **4 cas sur 15**. Même si son GAP moyen pénalisé, égal à **13,92 %**, est supérieur à celui de Claude Sonnet 4.6, Gemini 3 et DeepSeek, Grok 4 reste intéressant dans une logique interactive, lorsque plusieurs échanges permettent d'améliorer progressivement la solution.

Enfin, **ChatGPT 5.3** est le modèle le plus rapide, avec un temps de réponse moyen de **34,93 secondes**. Il présente aussi un bon taux de faisabilité, avec **14 solutions faisables sur 15**. Cependant, ses performances restent plus limitées en qualité de solution, en optimalité et en fiabilité, avec un GAP moyen pénalisé de **14,68 %**, seulement **2 solutions optimales sur 15** et **1 cas fiable sur 15**. Il peut donc produire rapidement une première piste de solution, mais celle-ci doit être systématiquement vérifiée.

Globalement, **Claude Sonnet 4.6** se distingue par la meilleure combinaison entre qualité de solution et optimalité. **Gemini 3** présente un bon GAP moyen et une

forte stabilité, tandis que **DeepSeek** obtient un GAP moyen relativement faible mais reste faible en optimalité et en fiabilité. **Grok 4** se démarque par sa fiabilité et sa capacité d'amélioration, alors que **ChatGPT 5.3** se distingue principalement par sa rapidité.des solutions.

4.8.7 Effet de la taille des instances

L'analyse des résultats montre que la taille des instances influence fortement les performances des LLM. Sur les petites instances, les modèles arrivent plus facilement à comprendre la structure du problème et à proposer une solution correcte ou proche de l'optimum.

Lorsque la taille du problème augmente, les performances se dégradent. Cette dégradation se manifeste par davantage d'erreurs de calcul, d'erreurs de faisabilité et d'hallucinations. Les modèles peuvent proposer des solutions présentées de manière cohérente, mais qui ne respectent pas toujours les contraintes du problème.

Les erreurs observées concernent principalement :

- le calcul incorrect de la fonction objectif ;
- l'annonce d'une valeur objectif différente de la valeur recalculée ;
- la sélection d'un nombre incorrect de sites ;
- la présence de doublons ou d'indices invalides ;
- l'affirmation qu'une solution est optimale alors qu'elle ne l'est pas ;
- la production de solutions faisables mais éloignées de l'optimum.

Cette tendance montre que les LLM peuvent donner une impression de maîtrise grâce à un raisonnement bien présenté, mais que cette maîtrise devient moins fiable lorsque le volume de données augmente. La vérification externe reste donc indispensable, surtout pour les grandes instances.

4.9 Résultats expérimentaux du problème *Set Covering Problem*

Cette section présente les résultats obtenus pour le *Set Covering Problem* dans la première phase expérimentale. Comme pour le p-median, l'analyse porte sur les résultats obtenus avec la configuration **avec rôle**.

Dans le SCP, une solution est faisable uniquement si toutes les lignes sont couvertes par les colonnes sélectionnées. Cette contrainte rend le problème particulièrement intéressant pour l'évaluation des LLM, car une réponse peut sembler correcte dans sa forme tout en étant infaisable après vérification.

L'analyse porte sur les 15 instances principales du benchmark SCP, de **scp41** à **scp65**. L'instance **c=12, s=25**, présente dans le fichier de résultats, est considérée comme une instance illustrative. Comme les cinq modèles y atteignent la solution optimale avec un GAP de 0 %, elle n'est pas intégrée aux moyennes afin de ne pas biaiser l'analyse.

4.9.1 Comparaison des GAP moyens

Dans cette analyse, le GAP moyen est calculé sur l'ensemble des **15 instances**. Lorsqu'une valeur numérique de GAP est disponible, elle est utilisée directement. En revanche, lorsqu'aucune valeur de GAP n'est renseignée, notamment en raison d'une solution infaisable, non vérifiable ou non exploitable, une valeur pénalisante de **100 %** est attribuée.

Table 4.16: Comparaison des GAP moyens pénalisés des LLM sur le problème SCP

Rang	**LLM**	**Average GAP pénalisé**
1	Claude Sonnet 4.6	4,61 %
2	Grok 4	26,66 %
3	ChatGPT 5.3	46,89 %
–	DeepSeek	100 %
–	Gemini 3	100 %

D'après le tableau 4.16, **Claude Sonnet 4.6** obtient le meilleur GAP moyen pénalisé sur le problème SCP, avec **4,61 %**. Ce résultat montre qu'il est le modèle dont les solutions sont globalement les plus proches des solutions de référence, tout en tenant compte des cas où aucune solution exploitable n'est disponible.

Grok 4 arrive en deuxième position avec un GAP moyen pénalisé de **26,66 %**. Ce résultat indique que le modèle produit plusieurs solutions de bonne qualité, mais qu'il reste moins performant que Claude Sonnet 4.6 en moyenne. Il conserve néanmoins un profil intéressant, notamment grâce à ses bons résultats en optimalité et en fiabilité.

ChatGPT 5.3 obtient un GAP moyen pénalisé de **46,89 %**. Ce résultat montre que certaines solutions peuvent être exploitables, mais que la performance globale reste plus irrégulière. Le modèle est donc moins compétitif que Claude Sonnet 4.6 et Grok 4 sur la qualité moyenne des solutions.

DeepSeek et **Gemini 3** obtiennent chacun un GAP moyen pénalisé de **100 %**. Cette valeur doit être interprétée comme une pénalisation liée à l'absence de solutions exploitables ou vérifiables, et non comme un écart numérique exact calculé à partir de solutions valides. Elle traduit donc une difficulté importante à produire des résultats utilisables sur le problème SCP.

Cette analyse montre que, pour le SCP, le GAP moyen pénalisé est plus pertinent qu'un GAP calculé uniquement sur les valeurs disponibles. Il permet de tenir compte à la fois de la qualité des solutions numériques obtenues et de l'absence de solutions exploitables. Dans ce problème, cette précaution est essentielle, car une solution de faible coût n'a aucune valeur si elle ne couvre pas correctement toutes les lignes.

4.9.2 Analyse de l'optimalité

L'optimalité est analysée à travers l'indicateur **Capability = Optimal**, qui indique si le modèle est finalement classé comme ayant obtenu une solution optimale après vérification complète des résultats.

Table 4.17: Nombre de solutions optimales obtenues par les LLM sur le problème SCP

LLM	**Capability = Optimal**
Grok 4	10 / 15
ChatGPT 5.3	3 / 15
Gemini 3	0 / 15
DeepSeek	0 / 15
Claude Sonnet 4.6	0 / 15

D'après le tableau 4.17, **Grok 4** domine clairement l'indicateur d'optimalité sur le problème SCP. Il obtient **10 solutions optimales sur 15** selon l'indicateur *Capability = Optimal*. Ce résultat montre que Grok 4 est le modèle le plus efficace pour atteindre l'optimum sur ce problème.

ChatGPT 5.3 arrive en deuxième position avec **3 solutions optimales sur 15**. Sa performance reste donc nettement inférieure à celle de Grok 4, mais elle montre que le modèle peut atteindre l'optimum dans certains cas.

Gemini 3, **DeepSeek** et **Claude Sonnet 4.6** n'obtiennent aucune solution optimale selon cet indicateur. Cela signifie qu'aucun de ces trois modèles n'a été classé comme optimal après vérification complète des solutions produites.

Le cas de **Claude Sonnet 4.6** doit cependant être nuancé. Même s'il n'atteint pas l'optimum sur les instances étudiées, il présente de bons résultats sur d'autres critères, notamment la faisabilité et la fiabilité. Il apparaît donc moins performant en optimalité stricte, mais plus robuste dans le respect des contraintes.

Globalement, **Grok 4** est le modèle le plus performant en termes d'optimalité sur le SCP, tandis que **ChatGPT 5.3** obtient quelques résultats optimaux. Les autres modèles ne parviennent pas à atteindre l'optimum selon l'indicateur *Capability = Optimal*.

4.9.3 Analyse de la faisabilité

La faisabilité est centrale dans le SCP. Une solution est faisable uniquement si toutes les lignes sont couvertes. Une solution de faible coût mais avec une couverture incomplète est donc infaisable.

Table 4.18: Comparaison de la faisabilité des solutions proposées par les LLM sur le SCP

Rang	**LLM**	**Feasibility = Yes**	**Percentage**
1	Claude Sonnet 4.6	15 / 15	100.00 %
2	Grok 4	11 / 15	73.33 %
3	ChatGPT 5.3	6 / 15	40.00 %
4	DeepSeek	1 / 15	6.67 %
5	Gemini 3	0 / 15	0.00 %

Claude Sonnet 4.6 est le modèle le plus robuste en faisabilité, avec **15 solutions faisables sur 15**. Même s'il n'atteint pas l'optimum, il respecte systématiquement les

contraintes de couverture.

Grok 4 obtient **11 solutions faisables sur 15**. Il combine donc de bons résultats en optimalité avec un niveau de faisabilité relativement élevé.

ChatGPT 5.3 obtient **6 solutions faisables sur 15**. Sa performance est plus irrégulière : il peut produire de bonnes solutions sur certaines instances, mais il génère aussi plusieurs solutions infaisables.

DeepSeek ne produit qu'une seule solution faisable, tandis que **Gemini 3** n'en produit aucune selon la colonne **Feasibility**. Ces résultats montrent de fortes limites pour ces deux modèles sur le respect des contraintes du SCP.

4.9.4 Analyse de la fiabilité

L'indicateur **Reliability** permet de vérifier si les affirmations du modèle correspondent aux résultats réellement contrôlés. Dans le SCP, cet indicateur est particulièrement important, car un modèle peut annoncer une solution correcte alors qu'une partie des lignes n'est pas couverte.

Table 4.19: Comparaison de la fiabilité des LLM sur le problème SCP

Rang	**LLM**	**Reliability = Yes**	**Percentage**
1	Claude Sonnet 4.6	15 / 15	100.00 %
2	Grok 4	10 / 15	66.67 %
3	ChatGPT 5.3	6 / 15	40.00 %
4	Gemini 3	0 / 15	0.00 %
5	DeepSeek	0 / 15	0.00 %

Claude Sonnet 4.6 est le modèle le plus fiable, avec **15 cas de Reliability = yes sur 15**. Ses affirmations sont donc cohérentes avec les résultats vérifiés sur toutes les instances principales.

Grok 4 arrive en deuxième position, avec **10 cas sur 15**. Il reste globalement fiable, même si certaines réponses nécessitent une vérification.

ChatGPT 5.3 obtient **6 cas de fiabilité positive**. Cela confirme son comportement irrégulier : il peut produire de bonnes réponses, mais il peut aussi annoncer des résultats non confirmés par le contrôle externe.

Gemini 3 et **DeepSeek** n'obtiennent aucun cas de fiabilité positive. Leurs réponses doivent donc être systématiquement contrôlées.

4.9.5 Amélioration au fil des itérations

L'indicateur **Improve Learning** permet d'observer si le modèle améliore sa solution après plusieurs interactions. Dans le SCP, cette amélioration peut correspondre à une meilleure couverture des contraintes, une correction de la solution ou une amélioration de la valeur objectif.

Table 4.20: Amélioration des LLM au fil des itérations sur le problème SCP

Rang	LLM	Improve Learning = Yes	Instances
1	Claude Sonnet 4.6	7 / 15	scp45, scp49, scp55, scp62, scp63, scp64, scp65
2	DeepSeek	3 / 15	scp41, scp45, scp59
3	Grok 4	2 / 15	scp59, scp62
3	Gemini 3	2 / 15	scp49, scp64
5	ChatGPT 5.3	0 / 15	Aucune

Claude Sonnet 4.6 est le modèle qui s'améliore le plus souvent, avec **7 cas d'amélioration sur 15**. Il ne trouve pas nécessairement l'optimum directement, mais il ajuste plus souvent sa réponse après interaction.

DeepSeek montre une amélioration dans **3 cas**, mais cette amélioration ne suffit pas toujours à produire une solution optimale ou fiable.

Grok 4 et **Gemini 3** montrent chacun **2 cas d'amélioration**. Pour Grok, cela s'explique en partie par le fait qu'il atteint déjà souvent l'optimum dès la première réponse. Pour Gemini, l'amélioration reste trop limitée pour compenser les faibles résultats en faisabilité.

ChatGPT 5.3 ne montre aucun cas d'amélioration dans les résultats observés sur le SCP.

4.9.6 Temps de réponse

Le temps de réponse permet de comparer la rapidité des modèles. Cet indicateur est utile pour un usage pratique, mais il doit toujours être interprété avec les autres indicateurs, car une réponse rapide peut être infaisable ou non fiable.

Table 4.21: Classement des LLM selon le temps de réponse sur le problème SCP

Rang	LLM	Average Response Time	Minimum Response Time	Maximum Response Time	Ranking
1	ChatGPT 5.3	52.60 s	10 s	200 s	Le plus rapide
2	Gemini 3	52.93 s	14 s	165 s	Rapide
3	DeepSeek	76.87 s	5 s	187 s	Moyen
4	Grok 4	128.93 s	12 s	450 s	Lent
5	Claude Sonnet 4.6	159.27 s	27 s	600 s	Le plus lent

ChatGPT 5.3 est le modèle le plus rapide, avec un temps moyen de **52.60 secondes**. Il est suivi de très près par **Gemini 3**, avec **52.93 secondes**.

DeepSeek occupe une position intermédiaire avec **76.87 secondes**. **Grok 4** et **Claude Sonnet 4.6** sont plus lents, avec respectivement **128.93 secondes** et **159.27 secondes**.

Ce classement montre encore une fois que la rapidité ne suffit pas. Les modèles les plus rapides ne sont pas forcément les plus fiables ni les plus faisables.

4.9.7 Analyse de la stabilité

L'indicateur **Stability** permet d'évaluer si le modèle conserve un comportement similaire lorsque le même prompt est soumis plusieurs fois. Un modèle stable produit généralement une même logique de réponse, tandis qu'un modèle instable peut modifier fortement sa solution ou sa méthode d'une tentative à l'autre.

Table 4.22: Comparaison de la stabilité des LLM sur le problème SCP

Rang	**LLM**	**Stability = Yes**	**Percentage**
1	ChatGPT 5.3	14 / 15	93.33 %
2	Grok 4	13 / 15	86.67 %
3	Gemini 3	12 / 15	80.00 %
4	Claude Sonnet 4.6	11 / 15	73.33 %
5	DeepSeek	5 / 15	33.33 %

ChatGPT 5.3 est le modèle le plus stable, avec **14 cas sur 15**. Cependant, cette stabilité ne signifie pas automatiquement que les solutions sont correctes. Un modèle peut répéter une logique similaire tout en gardant les mêmes erreurs.

Grok 4 présente également une bonne stabilité, avec **13 cas sur 15**. **Gemini 3** obtient **12 cas sur 15**, mais ses faibles résultats en faisabilité limitent l'intérêt pratique de cette stabilité.

Claude Sonnet 4.6 obtient **11 cas sur 15**. Sa stabilité est légèrement inférieure à celle de ChatGPT, Grok et Gemini, mais elle s'accompagne d'une faisabilité et d'une fiabilité très élevées.

DeepSeek est le modèle le moins stable, avec seulement **5 cas sur 15**.

4.9.8 Efficacité en nombre de requêtes

L'indicateur **Query Efficiency** indique le nombre de prompts nécessaires pour atteindre une solution optimale ou satisfaisante. Lorsque la solution n'est pas atteinte après trois requêtes, l'indicateur est noté **3+**.

Table 4.23: Efficacité des LLM en nombre de requêtes sur le problème SCP

LLM	**Solution reached in 1 query**	**Solution reached in 3 queries or less**	**3+ cases**
Grok 4	11 / 15	12 / 15	3 / 15
ChatGPT 5.3	8 / 15	8 / 15	7 / 15
Gemini 3	1 / 15	1 / 15	14 / 15
DeepSeek	0 / 15	1 / 15	14 / 15
Claude Sonnet 4.6	0 / 15	3 / 15	12 / 15

Grok 4 est le modèle le plus efficace en nombre de requêtes. Il atteint une solution en une seule requête dans **11 cas sur 15** et en trois requêtes ou moins dans **12 cas sur 15**.

ChatGPT 5.3 atteint une solution en une seule requête dans **8 cas sur 15**, mais présente aussi **7 cas notés 3+**. Ses performances sont donc plus irrégulières.

Gemini 3 et **DeepSeek** présentent une efficacité faible, avec **14 cas sur 15** notés **3+**. Même après plusieurs interactions, ces modèles n'atteignent généralement pas une solution satisfaisante.

Claude Sonnet 4.6 atteint une solution en trois requêtes ou moins dans **3 cas sur 15**, mais présente **12 cas 3+**. Il est donc moins efficace pour atteindre l'optimalité, même s'il reste très fiable et faisable.

4.9.9 Comparaison globale des LLM sur le SCP

Le tableau suivant synthétise les performances des cinq LLM selon les principaux indicateurs : GAP moyen, optimalité selon **Capability**, faisabilité, fiabilité, stabilité, amélioration au fil des itérations, efficacité en requêtes et temps moyen de réponse.

Table 4.24: Comparaison globale des LLM selon l'ensemble des indicateurs pour le SCP

LLM	Average GAP	Capability = Optimal	Feasibility = Yes	Reliability = Yes	Stability = Yes	Improve Learning	Query Efficiency ≤ 3	Average Response Time
ChatGPT 5.3	46,89 %	3 / 15	6 / 15	6 / 15	14 / 15	0 / 15	8 / 15	52,60 s
Gemini 3	100 %	0 / 15	0 / 15	0 / 15	12 / 15	2 / 15	1 / 15	52,93 s
DeepSeek	100 %	0 / 15	1 / 15	0 / 15	5 / 15	3 / 15	1 / 15	76,87 s
Grok 4	26,66 %	10 / 15	11 / 15	10 / 15	13 / 15	2 / 15	12 / 15	128,93 s
Claude Sonnet 4.6	4,61 %	0 / 15	15 / 15	15 / 15	11 / 15	7 / 15	3 / 15	159,27 s

La comparaison globale montre qu'aucun modèle ne domine l'ensemble des critères sur le problème SCP. Les performances varient fortement selon que l'on considère la qualité moyenne des solutions, l'optimalité, la faisabilité, la fiabilité ou encore le temps de réponse.

Claude Sonnet 4.6 obtient le meilleur GAP moyen, avec **4,61 %**. Il se distingue également par sa robustesse, puisqu'il produit **15 solutions faisables sur 15** et **15 cas fiables sur 15**. Même s'il n'obtient aucune solution optimale selon l'indicateur *Capability = Optimal*, il apparaît comme le modèle le plus stable et le plus sûr pour produire des solutions exploitables et respectant les contraintes.

Grok 4 est le modèle le plus performant en termes d'optimalité. Il obtient **10 solutions optimales sur 15** selon l'indicateur *Capability = Optimal*. Il présente aussi une bonne faisabilité, avec **11 solutions faisables sur 15**, ainsi qu'une bonne fiabilité, avec **10 cas fiables sur 15**. Son GAP moyen est de **26,66 %**, ce qui le place derrière Claude Sonnet 4.6 sur la qualité moyenne des solutions, mais devant les autres modèles. Il est également le plus efficace en nombre de requêtes, avec **12 cas sur 15** résolus en trois requêtes ou moins.

ChatGPT 5.3 est le modèle le plus rapide, avec un temps moyen de réponse de **52,60 secondes**. Il obtient **3 solutions optimales sur 15**, mais ses performances restent limitées en faisabilité, avec seulement **6 solutions faisables sur 15**. Son GAP moyen de **46,89 %** montre que ses solutions restent globalement éloignées des valeurs de référence. Il peut donc fournir rapidement une première piste, mais celle-ci doit être vérifiée avec attention.

Gemini 3 présente une bonne stabilité, avec **12 cas stables sur 15**, mais ses résultats sont faibles sur les autres indicateurs. Il n'obtient aucune solution optimale, aucune solution faisable et aucun cas fiable selon les résultats observés. Le GAP moyen de **100 %** traduit donc une performance pénalisée par l'absence de solutions exploitables.

DeepSeek présente également des performances limitées sur le SCP. Il n'obtient aucune solution optimale ni aucun cas fiable, et seulement **1 solution faisable sur 15**. Malgré **3 cas d'amélioration** au fil des interactions, cela ne suffit pas à compenser ses limites en faisabilité et en qualité de solution. Son GAP moyen de **100 %** confirme cette faiblesse globale.

Globalement, **Claude Sonnet 4.6** est le modèle le plus robuste pour le SCP grâce à sa faisabilité et sa fiabilité complètes, tandis que **Grok 4** est le plus performant en optimalité. **ChatGPT 5.3** se distingue surtout par sa rapidité, alors que **Gemini 3** et **DeepSeek** restent nettement moins convaincants sur ce problème.

4.9.10 Effet de la taille des instances

L'analyse des résultats montre que la taille et la complexité des instances influencent fortement le comportement des LLM. Sur les petites instances ou les cas simples, les modèles peuvent plus facilement produire une solution cohérente, voire optimale. Cela est visible sur l'instance illustrative **c=12, s=25**, où les cinq modèles atteignent l'optimum.

Lorsque la taille des instances augmente, les performances se dégradent. Dans le SCP, cette dégradation apparaît surtout à travers des erreurs de faisabilité. Les modèles peuvent sélectionner un ensemble de colonnes qui semble pertinent, mais qui ne couvre pas réellement toutes les lignes.

Les erreurs observées concernent principalement :

- la couverture incomplète de certaines lignes ;
- le mauvais calcul du coût total ;
- l'annonce d'une solution comme optimale alors qu'elle est infaisable ;
- la confusion entre solution de faible coût et solution faisable ;
- l'incapacité à vérifier correctement toutes les contraintes ;
- l'apparition de valeurs recalculées différentes des valeurs annoncées.

Cette tendance confirme que plus l'instance devient complexe, plus le risque d'hallucination augmente, notamment dans le calcul du coût, la vérification de la couverture et l'annonce du statut de la solution.

4.10 Résultats expérimentaux du problème ULS

Cette section présente les résultats obtenus pour le problème *Uncapacitated Lot Sizing Problem* (ULS) dans la première phase expérimentale. Comme pour le p-median et le *Set Covering Problem*, l'analyse porte uniquement sur la configuration **avec rôle**.

Le problème ULS consiste à déterminer les quantités à produire et les niveaux de stock sur plusieurs périodes, afin de satisfaire la demande tout en minimisant les coûts de production, de stockage et de lancement. Une solution est considérée comme faisable uniquement si elle respecte les équilibres de stock, satisfait toutes les demandes et respecte les contraintes liées aux décisions de production.

Les cinq LLM évalués sont **ChatGPT 5.3**, **Gemini 3**, **DeepSeek**, **Grok 4** et **Claude Sonnet 4.6**.

4.10.1 Comparaison des GAP moyens

Dans cette analyse, le GAP moyen est calculé sur l'ensemble des **15 instances**. Lorsqu'une valeur numérique de GAP est disponible, elle est utilisée directement. En revanche, lorsqu'aucune valeur de GAP n'est renseignée, notamment en raison d'une solution infaisable, non vérifiable ou non exploitable, une valeur pénalisante de **100 %** est attribuée.

Table 4.25: Comparaison des GAP moyens pénalisés des LLM sur le problème ULS

Rang	**LLM**	**Average GAP pénalisé**
1	Claude Sonnet 4.6	20,00 %
2	Grok 4	21,92 %
3	ChatGPT 5.3	38,08 %
4	Gemini 3	53,58 %
5	DeepSeek	77,63 %

D'après le tableau 4.25, **Claude Sonnet 4.6** obtient le meilleur GAP moyen pénalisé sur le problème ULS, avec **20,00 %**. Ce résultat montre qu'il est le modèle dont les solutions sont globalement les plus proches des solutions de référence, tout en tenant compte des cas où aucune valeur exploitable n'est disponible.

Grok 4 arrive en deuxième position avec un GAP moyen pénalisé de **21,92 %**. Ses performances sont donc proches de celles de Claude Sonnet 4.6. Cela montre que Grok 4 produit aussi des solutions compétitives sur le problème ULS, même si son écart moyen reste légèrement supérieur.

ChatGPT 5.3 obtient un GAP moyen pénalisé de **38,08 %**. Ce résultat indique que le modèle produit certaines solutions exploitables, mais que leur qualité reste globalement plus éloignée de l'optimum que celles de Claude Sonnet 4.6 et Grok 4.

Gemini 3 présente un GAP moyen pénalisé de **53,58 %**. Cette valeur traduit une difficulté plus importante à produire des solutions proches des valeurs de référence sur le problème ULS.

Enfin, **DeepSeek** obtient le GAP moyen pénalisé le plus élevé, avec **77,63 %**. Ce résultat montre que le modèle rencontre des difficultés importantes sur le problème ULS, aussi bien en qualité de solution qu'en capacité à produire des résultats exploitables.

Cette analyse montre que, pour le problème ULS, le GAP moyen pénalisé est plus pertinent qu'un GAP calculé uniquement sur les valeurs disponibles. Il permet de

tenir compte à la fois de la qualité des solutions numériques obtenues et de l'absence de solutions exploitables. Cette précaution est essentielle, car une solution ULS peut sembler intéressante en coût tout en étant inutilisable si les contraintes de demande, de production, de stock ou de setup ne sont pas respectées.

4.10.2 Analyse de l'optimalité

L'optimalité est analysée à travers deux indicateurs : **Optimality First Iteration**, qui indique si le modèle atteint l'optimum dès la première réponse, et **Capability**, qui indique si le modèle est finalement classé comme ayant obtenu une solution optimale.

Table 4.26: Nombre de solutions optimales obtenues par les LLM sur le problème ULS

LLM	**Optimality First Iteration**	**Capability = Optimal**
Claude Sonnet 4.6	12 / 15	12 / 15
Grok 4	10 / 15	8 / 15
Gemini 3	4 / 15	4 / 15
DeepSeek	2 / 15	2 / 15
ChatGPT 5.3	1 / 15	2 / 15

Claude Sonnet 4.6 est clairement le modèle le plus performant en optimalité. Il obtient **12 solutions optimales dès la première itération** et **12 solutions optimales selon Capability**. Cela montre qu'il est capable de résoudre correctement la majorité des instances ULS dès la première réponse.

Grok 4 arrive en deuxième position avec **10 solutions optimales dès la première itération** et **8 solutions optimales selon Capability**. Il montre donc une bonne capacité à trouver l'optimum, même si certaines solutions ne sont pas confirmées comme optimales après vérification complète.

Gemini 3 obtient **4 solutions optimales**, aussi bien dès la première itération que selon **Capability**. Sa performance est donc plus limitée, mais il reste meilleur que ChatGPT et DeepSeek sur cet indicateur.

DeepSeek obtient **2 solutions optimales**, tandis que **ChatGPT 5.3** n'obtient qu'une seule solution optimale dès la première itération et **2 solutions optimales selon Capability**.

Globalement, **Claude Sonnet 4.6** domine très largement l'analyse de l'optimalité sur le problème ULS.

4.10.3 Analyse de la faisabilité

La faisabilité est un indicateur essentiel pour ULS. Une solution est faisable uniquement si elle respecte les contraintes d'équilibre entre production, demande et stock sur toutes les périodes.

Table 4.27: Comparaison de la faisabilité des solutions proposées par les LLM sur le problème ULS

Rang	LLM	Feasibility = Yes	Percentage
1	Claude Sonnet 4.6	12 / 15	80.00 %
1	Grok 4	12 / 15	80.00 %
3	ChatGPT 5.3	10 / 15	66.67 %
4	Gemini 3	7 / 15	46.67 %
5	DeepSeek	4 / 15	26.67 %

Claude Sonnet 4.6 et **Grok 4** sont les modèles les plus performants en faisabilité, avec **12 solutions faisables sur 15** chacun. Cela montre qu'ils respectent plus souvent les contraintes du problème ULS que les autres modèles.

ChatGPT 5.3 obtient **10 solutions faisables sur 15**, ce qui reste correct. Cependant, plusieurs de ses solutions faisables restent éloignées de l'optimum, comme le montre son GAP moyen élevé.

Gemini 3 obtient **7 solutions faisables sur 15**. Sa performance est donc moyenne : il peut produire des solutions valides, mais il échoue sur plus de la moitié des instances.

DeepSeek est le modèle le moins performant en faisabilité, avec seulement **4 solutions faisables sur 15**. Cela montre une difficulté importante à respecter les contraintes du ULS.

Cette analyse montre que Claude et Grok sont les plus robustes pour produire des solutions faisables, tandis que DeepSeek présente de fortes limites sur ce point.

4.10.4 Analyse de la fiabilité

La **Reliability** permet de vérifier si ce que le LLM affirme correspond réellement aux résultats contrôlés. Pour ULS, cet indicateur est très important, car un modèle peut annoncer une solution optimale ou correcte alors que la valeur recalculée ou la faisabilité montrent le contraire.

Table 4.28: Comparaison de la fiabilité des LLM sur le problème ULS

Rang	LLM	Reliability = Yes	Percentage
1	Claude Sonnet 4.6	12 / 15	80.00 %
2	Grok 4	11 / 15	73.33 %
3	ChatGPT 5.3	5 / 15	33.33 %
3	Gemini 3	5 / 15	33.33 %
5	DeepSeek	2 / 15	13.33 %

Claude Sonnet 4.6 est le modèle le plus fiable, avec **12 cas de Reliability = yes sur 15**. Cela signifie que ses affirmations correspondent le plus souvent aux résultats vérifiés.

Grok 4 arrive juste derrière avec **11 cas sur 15**. Il montre donc également une bonne cohérence entre ce qu'il annonce et ce qui est réellement vérifié.

ChatGPT 5.3 et **Gemini 3** obtiennent chacun **5 cas de reliability positive**. Cela montre que leurs réponses doivent être vérifiées avec prudence, même lorsqu'elles semblent bien structurées.

DeepSeek obtient seulement **2 cas de Reliability = yes**, ce qui confirme ses difficultés sur le problème ULS.

Cette analyse montre que Claude et Grok sont les modèles les plus fiables, tandis que ChatGPT, Gemini et DeepSeek nécessitent une vérification systématique.

4.10.5 Amélioration au fil des itérations

L'indicateur **Improve Learning** permet de voir si le modèle améliore sa réponse après une nouvelle interaction ou une correction. Dans le tableau expérimental, seuls les "yes" explicites ont été comptés comme des cas d'amélioration. Certaines cellules contiennent des valeurs numériques dans cette colonne ; elles ne sont pas comptées comme des "yes", car elles ne correspondent pas explicitement à une amélioration validée.

Table 4.29: Amélioration des LLM au fil des itérations sur le problème ULS

Rang	**LLM**	**Improve Learning = Yes**	**Instances**
1	Claude Sonnet 4.6	1 / 15	120.2
2	ChatGPT 5.3	0 / 15	Aucune
2	Gemini 3	0 / 15	Aucune
2	DeepSeek	0 / 15	Aucune
2	Grok 4	0 / 15	Aucune

L'indicateur **Improve Learning** est peu marqué pour le problème ULS. Seul **Claude Sonnet 4.6** présente un cas explicite d'amélioration, sur l'instance **120.2**.

Les autres modèles ne montrent aucun cas d'amélioration clairement indiqué dans le tableau. Cela suggère que, pour ULS, les interactions supplémentaires ne permettent pas toujours aux modèles de corriger leurs erreurs ou d'améliorer leurs solutions.

Ce résultat est important, car il montre que les erreurs sur ULS ne sont pas toujours facilement corrigées par simple relance ou reformulation du prompt.

4.10.6 Temps de réponse

Le temps de réponse permet de comparer la rapidité des modèles. Comme pour les autres problèmes, cet indicateur doit être interprété avec prudence, car une réponse rapide peut être infaisable ou éloignée de l'optimum.

Table 4.30: Classement des LLM selon le temps de réponse sur le problème ULS

Rang	LLM	Average Response Time	Minimum Response Time	Maximum Response Time	Ranking
1	ChatGPT 5.3	39.60 s	15 s	120 s	Le plus rapide
2	Gemini 3	44.60 s	15 s	124 s	Rapide
3	DeepSeek	58.53 s	18 s	180 s	Moyen
4	Claude Sonnet 4.6	143.53 s	30 s	500 s	Lent
5	Grok 4	219.53 s	22 s	600 s	Le plus lent

ChatGPT 5.3 est le modèle le plus rapide, avec un temps moyen de **39.60 secondes**. Il est suivi par **Gemini 3**, avec **44.60 secondes** en moyenne.

DeepSeek occupe une position intermédiaire avec un temps moyen de **58.53 secondes**.

Claude Sonnet 4.6 est plus lent, avec un temps moyen de **143.53 secondes**, mais il obtient les meilleurs résultats en optimalité, faisabilité et fiabilité.

Grok 4 est le modèle le plus lent, avec un temps moyen de **219.53 secondes**. Malgré cette lenteur, il reste performant en faisabilité et en fiabilité.

Cette analyse montre encore une fois que la rapidité ne garantit pas la qualité. ChatGPT est le plus rapide, mais Claude est le plus performant en qualité de solution.

4.10.7 Analyse de la stabilité

La stabilité permet d'évaluer si le modèle garde un comportement cohérent lorsque le prompt est répété ou lorsqu'une interaction similaire est effectuée.

Table 4.31: Comparaison de la stabilité des LLM sur le problème ULS

Rang	LLM	Stability = Yes	Percentage
1	Claude Sonnet 4.6	15 / 15	100.00 %
2	Gemini 3	14 / 15	93.33 %
2	Grok 4	14 / 15	93.33 %
4	ChatGPT 5.3	12 / 15	80.00 %
5	DeepSeek	8 / 15	53.33 %

Claude Sonnet 4.6 est le modèle le plus stable, avec **15 cas de Stability = yes sur 15**. Il conserve donc un comportement cohérent sur toutes les instances observées.

Gemini 3 et **Grok 4** présentent également une très bonne stabilité, avec **14 cas sur 15** chacun.

ChatGPT 5.3 obtient **12 cas de stabilité positive**, ce qui reste correct.

DeepSeek est le modèle le moins stable, avec **8 cas sur 15**. Cela indique une plus grande variabilité dans ses réponses.

Cependant, comme pour les autres problèmes, la stabilité ne signifie pas nécessairement que la solution est correcte. Un modèle peut être stable tout en répétant une erreur.

4.10.8 Efficacité en nombre de requêtes

La **Query Efficiency** indique le nombre de requêtes nécessaires pour atteindre une solution optimale ou satisfaisante. Lorsque la solution n'est pas atteinte après trois requêtes, l'indicateur est noté **3+**.

Table 4.32: Efficacité des LLM en nombre de requêtes sur le problème ULS

LLM	**Solution reached in 1 query**	**Solution reached in 3 queries or less**	**3+ cases**
Claude Sonnet 4.6	12 / 15	12 / 15	3 / 15
Grok 4	10 / 15	10 / 15	5 / 15
Gemini 3	4 / 15	4 / 15	11 / 15
DeepSeek	2 / 15	2 / 15	13 / 15
ChatGPT 5.3	1 / 15	1 / 15	14 / 15

Claude Sonnet 4.6 est le modèle le plus efficace en nombre de requêtes. Il atteint une solution en une seule requête dans **12 cas sur 15**.

Grok 4 arrive en deuxième position avec **10 cas sur 15**. Il atteint souvent une solution directement, mais reste moins efficace que Claude.

Gemini 3 atteint une solution en une requête dans **4 cas sur 15**, tandis que **DeepSeek** y arrive dans seulement **2 cas**.

ChatGPT 5.3 est le moins efficace selon cet indicateur, avec seulement **1 solution atteinte en une requête** et **14 cas notés 3+**.

Cette analyse montre que Claude est le modèle le plus efficace pour atteindre rapidement une solution satisfaisante sur ULS, tandis que ChatGPT nécessite souvent plusieurs interactions sans forcément améliorer le résultat.

4.10.9 Comparaison globale des LLM sur le ULS

Le tableau suivant synthétise les performances des cinq LLM selon les principaux indicateurs : GAP moyen pénalisé, optimalité selon **Capability**, faisabilité, fiabilité, stabilité, amélioration au fil des itérations, efficacité en requêtes et temps moyen de réponse.

Table 4.33: Comparaison globale des LLM selon l'ensemble des indicateurs pour le problème ULS

LLM	**Average GAP pénalisé**	**Capability = Optimal**	**Feasibility = Yes**	**Reliability = Yes**	**Stability = Yes**	**Improve Learning**	**Query Efficiency ≤ 3**	**Average Response Time**
ChatGPT 5.3	38,08 %	2 / 15	10 / 15	5 / 15	12 / 15	0 / 15	1 / 15	39,60 s
Gemini 3	53,58 %	4 / 15	7 / 15	5 / 15	14 / 15	0 / 15	4 / 15	44,60 s
DeepSeek	77,63 %	2 / 15	4 / 15	2 / 15	8 / 15	0 / 15	2 / 15	58,53 s
Grok 4	21,92 %	8 / 15	12 / 15	11 / 15	14 / 15	0 / 15	10 / 15	219,53 s
Claude Sonnet 4.6	20,00 %	12 / 15	12 / 15	12 / 15	15 / 15	1 / 15	12 / 15	143,53 s

La comparaison globale montre que **Claude Sonnet 4.6** est le modèle le plus performant sur le problème ULS. Il obtient le meilleur GAP moyen pénalisé, avec **20,00 %**, ainsi que le plus grand nombre de solutions optimales selon *Capability*, avec **12 solutions optimales sur 15**. Il présente également la meilleure stabilité, avec **15 cas stables sur 15**, la meilleure fiabilité, avec **12 cas fiables sur 15**, et la meilleure efficacité en nombre de requêtes, avec **12 cas sur 15** résolus en trois requêtes ou moins. Il apparaît donc comme le modèle le plus robuste et le plus fiable sur ce problème.

Grok 4 arrive en deuxième position. Il obtient un GAP moyen pénalisé de **21,92 %**, très proche de celui de Claude Sonnet 4.6. Il atteint également **8 solutions optimales sur 15**, produit **12 solutions faisables sur 15** et présente une bonne fiabilité, avec **11 cas fiables sur 15**. Ses résultats montrent qu'il est performant sur le ULS, même si son temps moyen de réponse reste élevé, avec **219,53 secondes**.

ChatGPT 5.3 est le modèle le plus rapide, avec un temps moyen de réponse de **39,60 secondes**. Il produit **10 solutions faisables sur 15**, ce qui reste correct. Cependant, son GAP moyen pénalisé de **38,08 %**, ses **2 solutions optimales sur 15** et sa faible efficacité en nombre de requêtes montrent que ses performances restent limitées. Il peut donc être utile pour générer rapidement une première proposition, mais celle-ci doit être fortement vérifiée.

Gemini 3 présente une bonne stabilité, avec **14 cas stables sur 15**, et obtient **4 solutions optimales sur 15**. Cependant, son GAP moyen pénalisé de **53,58 %** et ses **7 solutions faisables sur 15** montrent que ses résultats restent insuffisamment robustes sur le problème ULS. Il est donc stable dans sa manière de répondre, mais pas toujours fiable ni proche de l'optimum.

DeepSeek présente les résultats les plus faibles sur la majorité des indicateurs. Il obtient le GAP moyen pénalisé le plus élevé, avec **77,63 %**, seulement **4 solutions faisables sur 15**, **2 solutions optimales sur 15** et **2 cas fiables sur 15**. Ces résultats montrent qu'il rencontre des difficultés importantes sur le problème ULS, aussi bien en qualité de solution qu'en respect des contraintes.

Globalement, **Claude Sonnet 4.6** se distingue comme le modèle le plus robuste sur le ULS, suivi par **Grok 4**, qui présente des résultats proches mais avec un temps de réponse plus élevé. **ChatGPT 5.3** se démarque principalement par sa rapidité, tandis que **Gemini 3** reste stable mais moins fiable. **DeepSeek** est le modèle le moins performant sur ce problème.

4.10.10 Effet de la taille des instances

L'analyse par taille montre une dégradation nette des performances lorsque les instances deviennent plus longues ou plus complexes.

Table 4.34: Effet de la taille des instances sur les performances des LLM pour le problème ULS

Groupe	Optimality First Iteration	Capability = Optimal	Feasibility = Yes	Reliability = Yes	Reported Average GAP
Petites instances	16 / 25	15 / 25	20 / 25	17 / 25	2.65 %
Instances moyennes	9 / 25	7 / 25	12 / 25	7 / 25	52.07 %
Grandes instances	4 / 25	6 / 25	13 / 25	11 / 25	51.17 %

Les petites instances sont clairement mieux résolues. Les modèles obtiennent plus souvent des solutions optimales, faisables et fiables. Le GAP moyen global renseigné est seulement de **2.65 %**, ce qui montre que les réponses sont proches de l'optimum.

À partir des instances moyennes, les performances chutent fortement. Le nombre de solutions optimales diminue, la faisabilité baisse et le GAP moyen augmente fortement pour atteindre **52.07 %**. Cela montre que les modèles commencent à rencontrer des difficultés importantes lorsque l'horizon de planification devient plus long.

Sur les grandes instances, le nombre de solutions optimales dès la première itération baisse encore. La faisabilité remonte légèrement grâce aux bonnes performances de Claude et Grok, mais les erreurs restent nombreuses. Le GAP moyen global reste très élevé, avec **51.17 %**.

Cette tendance montre que l'augmentation de la taille des instances accentue les hallucinations et les erreurs. Dans le cas du ULS, ces erreurs concernent surtout :

- le calcul incorrect du coût total ;
- le non-respect de l'équilibre entre production, demande et stock ;
- l'annonce d'une solution optimale alors qu'elle ne l'est pas ;
- la confusion entre solution faisable et solution optimale ;
- les écarts entre la valeur annoncée par le modèle et la valeur recalculée ;
- la difficulté à maintenir une cohérence sur un horizon de planification plus long.

Cela confirme que les LLM ont plus de difficulté à gérer les problèmes séquentiels longs. ULS demande une cohérence période par période, et cette contrainte devient plus difficile à respecter lorsque le nombre de périodes augmente.

4.11 Conclusion

Ce chapitre a présenté la première phase expérimentale de ce travail, consacrée à l'évaluation des LLM comme solveurs autonomes pour des problèmes d'optimisation combinatoire. L'objectif était d'analyser leur capacité à produire des solutions exploitables sans méthode guidée, solveur externe ou code de vérification fourni au départ.

Les résultats obtenus sur les trois problèmes étudiés, à savoir le p-median, le *Set Covering Problem* et le *Uncapacitated Lot Sizing Problem*, montrent que les LLM peuvent générer des réponses structurées et parfois des solutions de bonne qualité. Cependant, leurs performances restent très variables selon le modèle, le problème traité et la taille des instances.

Pour le problème du p-median, **Claude Sonnet 4.6** obtient les meilleurs résultats globaux, notamment grâce à son faible GAP moyen pénalisé et à son nombre élevé de solutions optimales. **Gemini 3** et **DeepSeek** présentent également des GAP moyens intéressants, mais leurs performances restent plus faibles en fiabilité et en optimalité. **Grok 4** se distingue surtout par sa fiabilité et sa capacité d'amélioration au fil des interactions.

Pour le *Set Covering Problem*, les résultats montrent que la faisabilité est un critère essentiel. Une solution de faible coût ne peut pas être acceptée si elle ne couvre pas toutes les lignes. Sur ce problème, **Grok 4** est le plus performant en termes d'optimalité, tandis que **Claude Sonnet 4.6** apparaît comme le modèle le plus robuste en faisabilité, en fiabilité et en GAP moyen pénalisé.

Pour le problème ULS, **Claude Sonnet 4.6** obtient les meilleurs résultats globaux. Il se distingue par son GAP moyen pénalisé, son nombre élevé de solutions optimales, sa faisabilité, sa fiabilité et sa stabilité. **Grok 4** présente également de bonnes performances, mais avec un temps de réponse plus élevé. Ce problème met particulièrement en évidence la difficulté des LLM à maintenir la cohérence entre production, demande, stock et setup sur plusieurs périodes.

De manière générale, aucun modèle ne domine parfaitement sur tous les critères. **Claude Sonnet 4.6** ressort comme le modèle le plus robuste, tandis que **Grok 4** se distingue par sa fiabilité et sa capacité d'amélioration. **ChatGPT 5.3** est souvent le plus rapide, mais ses résultats confirment que la rapidité ne garantit pas la qualité. **Gemini 3** présente une bonne stabilité dans certains cas, mais une fiabilité limitée. **DeepSeek** obtient des résultats plus contrastés, avec certaines solutions proches de l'optimum mais une faible fiabilité globale.

Cette phase montre également que la taille des instances influence fortement les performances. Les petites instances sont plus facilement traitées par les LLM, tandis que les grandes instances entraînent davantage d'erreurs de calcul, d'incohérences et de violations de contraintes. Ainsi, une réponse bien formulée ne garantit pas nécessairement une solution correcte.

En conclusion, les LLM ne peuvent pas être considérés comme des substituts fiables aux méthodes classiques d'optimisation lorsqu'ils sont utilisés comme solveurs autonomes. Ils peuvent néanmoins jouer un rôle utile comme outils d'aide initiale, en proposant des pistes de solution, des formulations ou des explications. Leur utilisation doit cependant toujours être accompagnée d'une vérification externe afin de contrôler la faisabilité, recalculer la fonction objectif et mesurer le GAP réel. Ces limites justifient la phase suivante, dans laquelle les LLM seront étudiés comme assistants guidés plutôt que comme solveurs entièrement autonomes.

Conclusion générale

Ce mémoire d'ingéniorat s'est intéressé au potentiel de l'intelligence artificielle générative, et plus particulièrement des grands modèles de langage (*Large Language Models*, LLM), dans un contexte technique nécessitant du raisonnement structuré, de la rigueur mathématique et une vérification objective des résultats. L'objectif principal était d'évaluer dans quelle mesure les réponses produites par ces modèles peuvent être considérées comme fiables, exploitables et utiles dans une démarche d'aide à la décision.

L'optimisation combinatoire a été retenue comme terrain d'application, car elle permet de vérifier objectivement les réponses générées à travers plusieurs critères : faisabilité, valeur de la fonction objectif, GAP moyen pénalisé, stabilité, fiabilité et temps de réponse. Trois problèmes classiques ont été étudiés : le p-median, le *Set Covering Problem* et l'*Uncapacitated Lot Sizing Problem*. Ces problèmes offrent des structures différentes et permettent d'évaluer les LLM dans des situations variées de localisation, de couverture et de planification de production.

Les premiers chapitres ont permis de poser les bases théoriques nécessaires à l'étude. Ils ont présenté les notions d'intelligence artificielle, de machine learning, de deep learning, d'IA générative et de LLM, puis ont introduit les principes du *prompt engineering*. Ces éléments ont montré que les LLM peuvent produire du texte, du code, des raisonnements structurés et des explications, mais qu'ils restent sensibles à la formulation du prompt et ne fournissent pas de garanties formelles sur la validité de leurs réponses.

Un framework de *prompt engineering* adapté à l'optimisation combinatoire a ensuite été proposé. Celui-ci repose notamment sur l'utilisation de l'anglais, l'attribution d'un rôle d'expert, l'interdiction de reproduire des solutions optimales connues, la décomposition de la tâche, la spécification du format de sortie et la mesure du temps de réponse. Ce cadre a permis de construire des prompts structurés, comparables et reproductibles.

L'évaluation expérimentale a porté sur cinq modèles : **ChatGPT 5.3**, **Gemini 3**, **DeepSeek**, **Grok 4** et **Claude Sonnet 4.6**. Les résultats ont mis en évidence des différences importantes entre les modèles. **Claude Sonnet 4.6** s'est distingué par sa robustesse globale, notamment en termes de qualité des solutions, de faisabilité et de stabilité. **Grok 4** a également montré de bonnes performances sur certains critères, en particulier l'optimalité et la fiabilité dans plusieurs cas. **ChatGPT 5.3** a produit des réponses structurées et rapides, mais avec des limites en qualité et en fiabilité. **Gemini 3** a montré une certaine stabilité, tandis que **DeepSeek** a obtenu des résultats plus contrastés selon les problèmes étudiés.

Les résultats confirment que les LLM ne peuvent pas être considérés, à ce stade, comme des solveurs autonomes fiables pour des problèmes d'optimisation combinatoire. Ils peuvent proposer des solutions, expliquer une démarche, générer du code ou suggérer des heuristiques, mais leurs réponses doivent être systématiquement vérifiées. Une solution bien rédigée peut être infaisable, une valeur objective peut être mal calculée, et une affirmation d'optimalité peut être incorrecte. La vérification externe par code, solveur ou procédure de contrôle reste donc indispensable.

Ce travail met également en évidence l'effet de la taille et de la complexité des instances. Sur les petites instances, certains modèles parviennent à produire des solutions cor-

rectes ou proches des références. En revanche, lorsque la taille du problème augmente, les erreurs deviennent plus fréquentes, notamment dans le respect des contraintes, le calcul de la fonction objectif et la cohérence des décisions proposées.

La principale contribution de ce mémoire est donc de proposer une évaluation structurée du potentiel des LLM dans un cadre technique vérifiable. Le travail montre que ces modèles peuvent jouer un rôle utile comme outils d'assistance : ils peuvent aider à comprendre un problème, structurer une formulation, proposer une première solution, générer du code ou accompagner une démarche de résolution. En revanche, leur utilisation comme solveurs autonomes reste limitée, car ils ne fournissent pas de garanties suffisantes en termes de faisabilité, d'optimalité et de fiabilité.

Ce travail présente toutefois certaines limites. Les résultats dépendent des modèles testés, des instances retenues et des prompts utilisés. Les performances des LLM peuvent également évoluer avec le temps, car ces modèles sont régulièrement mis à jour. De plus, l'étude porte sur des problèmes classiques et déterministes, alors que des problèmes industriels plus complexes, dynamiques ou incertains pourraient révéler d'autres difficultés.

Plusieurs perspectives peuvent être envisagées. Une première piste consiste à étudier les LLM dans un cadre plus guidé, en les associant à des heuristiques, des métaheuristiques ou des solveurs spécialisés. Une deuxième perspective serait de développer des architectures hybrides combinant LLM, vérificateurs automatiques et solveurs, afin de créer une boucle de correction et d'amélioration progressive des solutions. Enfin, l'étude pourrait être étendue à des problèmes plus complexes, notamment dans le domaine du routage, de la planification industrielle ou de l'aide multicritère à la décision.

En conclusion, l'intelligence artificielle générative offre un potentiel réel pour l'ingénierie et l'aide à la décision, mais ce potentiel doit être abordé avec rigueur. Les LLM ne doivent pas être considérés comme des remplaçants directs des méthodes classiques d'optimisation. Ils doivent plutôt être vus comme des assistants capables d'accompagner l'utilisateur dans la modélisation, l'explication et la génération de solutions candidates, au sein d'un processus contrôlé, vérifiable et hybride.

Annexe A

Prompts utilisés dans les expérimentations

A.1 Prompt du problème *p*-median

Cette annexe présente les prompts utilisés pour le problème du p-median dans les deux configurations expérimentales : avec spécification de rôle et sans spécification de rôle. Ces prompts ont été conçus afin d'évaluer l'impact du *role prompting* sur la qualité des réponses produites par les LLM.

A.1.1 Prompt avec spécification de rôle

Listing A.1: Prompt du problème p-median avec spécification de rôle

```
1-Prompt : WITH role specification

You are an expert in combinatorial optimization and mathematical
    programming.

Solve the **pmed1** instance from the OR-Library benchmark and
   explain your reasoning step by step.

Important constraint:
- You are allowed to use the benchmark instance and its input
   data.
- You are NOT allowed to directly use, recall, or reproduce
   known optimal solutions or published results associated with
   this benchmark.
- The solution must be obtained by explicitly performing the
   necessary modeling, reasoning, and calculations based solely
   on the provided data.

Context / Data:

- Instance: pmed1 (OR-Library format)
- Expected file format:
  - First line: n m p
```

```
  - Followed by n lines, each containing m distance values d(i,j
    )

If the exact content of the file is not provided, clearly
   explain how you would read and parse the pmed1 file according
   to the OR-Library format.

Tasks:

1) Data understanding and parsing
   - Describe how the pmed1 file is parsed
   - Report the values of n, m, and p
   - Briefly describe the structure of the distance matrix and
      provide a short illustrative excerpt

2) Mathematical formulation
   - Formulate the p-median problem for this instance
   - Define the decision variables
   - State the objective function
   - List all constraints and specify their binary nature when
      applicable

3) Solution
   - Compute the solution based on your formulation
   - Provide the selected facility locations (indices or labels)
   - Provide the client-to-facility assignment in a table (
      client | assigned facility | distance)
   - Report the total minimum distance

4) Solution method
   - Clearly state the solution approach used (exact method,
      heuristic, or metaheuristic)
   - If an exact method is used, describe the resolution process
      and stopping criteria
   - If a heuristic or local search is used, describe the
      algorithm (initialization, neighborhood moves, stopping
      criterion)

5) Performance information
   - Report the algorithmic computation time or number of
      iterations
   - Describe convergence behavior if applicable

6) Justification and discussion
   - Explain why the chosen method is suitable for the pmed1
      instance
   - Discuss limitations, robustness, and possible alternatives
      for larger instances

7) Response time measurement
```

```
   - Explicitly report the time required to generate the
      complete solution and explanation (in seconds)
   - Clearly distinguish this response time from the algorithmic
      computation time

8) Presentation
   - Present the final assignment in a clear table
   - Summarize the total distance in one line at the end
```

A.1.2 Prompt sans spécification de rôle

Listing A.2: Prompt du problème p-median sans spécification de rôle

```
2-Prompt : WITHOUT role specification

Solve the **pmed1** instance from the OR-Library benchmark and
   explain your steps in detail.

Important constraint:
- The benchmark data may be used as input.
- Known optimal solutions or published benchmark results must
   not be directly reused.
- The solution must be computed through explicit modeling,
   reasoning, and calculations.

Context / Data:

- Instance: pmed1 (OR-Library format)
- Expected file format:
  - First line: n m p
  - Followed by n lines, each containing m distance values d(i,j
     )

If the exact content of the file is not provided, explain how
   you would read and parse the pmed1 file according to the OR-
   Library format.

Tasks:

1) Describe how the pmed1 file is parsed and report the values
   of n, m, and p

2) Formulate the p-median problem:
   - decision variables
   - objective function
   - constraints

3) Compute and report a solution:
   - selected facility locations
   - client-to-facility assignment (table)
   - total distance
```

```
4) Describe the solution method used

5) Provide performance-related information:
   - computation time or iteration count if applicable

6) Briefly justify the method and discuss limitations and
   robustness

7) Response time measurement:
   - Report the time required to generate the full answer (in
     seconds)

8) Present the results clearly, including a final summary of the
    total distance
```

A.2 Prompt du *Set Covering Problem*

Cette section présente les prompts utilisés pour le *Set Covering Problem* dans les deux configurations expérimentales : avec spécification de rôle et sans spécification de rôle. L'objectif est d'évaluer l'effet du *role prompting* sur la capacité des LLM à formuler, résoudre et vérifier un problème de couverture par ensembles.

A.2.1 Prompt avec spécification de rôle

Listing A.3: Prompt du Set Covering Problem avec spécification de rôle

```
1-Prompt : WITH role specification

You are an expert in combinatorial optimization and mathematical
    programming.

Solve the **scp41** instance from the OR-Library benchmark (Set
   Covering Problem) and explain your reasoning step by step.

Important constraints:
- You are allowed to use the benchmark instance and its input
   data.
- You are NOT allowed to directly use, recall, or reproduce
   known optimal solutions or published results associated with
   this benchmark.
- The solution must be obtained by explicitly performing the
   necessary modeling, reasoning, and calculations based solely
   on the provided data.

Context / Data:

- Instance: scp41 (OR-Library - Set Covering Problem)
- Expected file format (OR-Library SCP format):
```

```
  - First line: m n
    (m = number of rows / elements to be covered, n = number of
       columns / sets)
  - Second line: c1 c2 ... cn (costs of each set)
  - Then m lines, each describing:
      k  j1 j2 ... jk
    where the current row is covered by columns j1, j2, ..., jk

If the exact content of the file is not provided, clearly
   explain how you would read and parse the scp41 file according
   to the OR-Library SCP format.

Tasks:

1) Data understanding and parsing
   - Describe how the scp41 file is parsed
   - Report the values of m and n
   - Briefly describe the structure of the covering data and
      provide a short illustrative excerpt
   - Explain how the binary coverage matrix A is reconstructed
      from the file

2) Mathematical formulation
   - Formulate the Set Covering Problem for this instance
   - Define the binary decision variables
   - State the objective function
   - List all constraints

3) Solution
   - Compute a solution based on your formulation
   - Provide the selected sets (columns) indices
   - Provide coverage information in a table (row | covered by
      selected sets)
   - Report the total cost of the selected sets

4) Solution method
   - Clearly state the solution approach used (exact MILP,
      greedy heuristic, local search, or metaheuristic)
   - If an exact method is used, describe the branch-and-bound /
       MILP solving process and stopping criteria
   - If a heuristic is used, describe the algorithm (
      initialization, selection rule, improvement moves, stopping
       criterion)

5) Performance information
   - Report the algorithmic computation time or number of
      iterations
   - Describe convergence behavior if applicable

6) Justification and discussion
```

```
   - Explain why the chosen method is suitable for the scp41
     instance
   - Discuss limitations, robustness, and possible alternatives
     for larger SCP instances

7) Response time measurement
   - Explicitly report the time required to generate the
     complete solution and explanation (in seconds)
   - Clearly distinguish this response time from the algorithmic
     computation time

8) Presentation
   - Present the final selected columns clearly
   - Summarize the total cost in one line at the end
```

A.2.2 Prompt sans spécification de rôle

Listing A.4: Prompt du Set Covering Problem sans spécification de rôle

```
2-Prompt : WITHOUT role specification

Solve the **scp41** instance from the OR-Library benchmark (Set
   Covering Problem) and explain your steps in detail.

Important constraints:
- The benchmark data may be used as input.
- Known optimal solutions or published benchmark results must
   not be directly reused.
- The solution must be computed through explicit modeling,
   reasoning, and calculations.

Context / Data:

- Instance: scp41 (OR-Library - Set Covering Problem format)
- Expected file format:
  - First line: m n
    (m = number of rows to be covered, n = number of columns /
       sets)
  - Second line: c1 c2 ... cn (cost of each column)
  - Then m lines:
      k  j1 j2 ... jk
    where row i is covered by columns j1, j2, ..., jk

If the exact content of the file is not provided, explain how
   you would read and parse the scp41 file according to the OR-
   Library SCP format.

Tasks:

1) Describe how the scp41 file is parsed and report the values
   of m and n
```

```
   - Explain how the binary coverage matrix A is reconstructed

2) Formulate the Set Covering Problem:
   - decision variables
   - objective function
   - constraints

3) Compute and report a solution:
   - selected columns (sets)
   - coverage table (row | covered by which selected column)
   - total cost

4) Describe the solution method used

5) Provide performance-related information:
   - computation time or iteration count if applicable

6) Briefly justify the method and discuss limitations and
   robustness

7) Response time measurement:
   - Report the time required to generate the full answer (in
      seconds)

8) Present the results clearly, including a final summary of the
    total cost
```

A.3 Prompt du problème *Uncapacitated Lot Sizing*

Cette section présente les prompts utilisés pour le problème *Uncapacitated Lot Sizing Problem* (ULS), dans les deux configurations expérimentales : avec spécification de rôle et sans spécification de rôle. Ces prompts ont été conçus afin d'évaluer la capacité des LLM à interpréter une instance de planification de production, à formuler le problème mathématiquement, puis à proposer un plan de production faisable et vérifiable.

A.3.1 Prompt avec spécification de rôle

Listing A.5: Prompt du problème ULS avec spécification de rôle

```
1-Prompt : WITH role specification

You are an expert in combinatorial optimization.

Solve the provided benchmark instance of the Uncapacitated Lot
   Sizing Problem (ULS) and explain the modeling and reasoning
   process.

Important constraints:
```

```
You are allowed to use the benchmark instance and its input data
   .
You are NOT allowed to directly recall, reproduce, or rely on
   previously published optimal solutions associated with this
   benchmark.
The solution must be obtained by explicitly performing the
   modeling and reasoning using only the provided data.

Instance format:
Line 1: Number of planning periods.
Line 2: Demand for each period.
Line 3: Inventory holding cost per unit carried from period t to
    t+1.
Line 4: Fixed setup cost incurred if production occurs in period
    t.
Line 5: Unit production cost.

Tasks:

Data understanding and parsing
Explain how the instance file is parsed.
Report the value of T.
Present the demand vector.
Present the holding cost vector.
Present the setup cost vector.
Present the unit production cost.
Briefly interpret what these parameters represent in the context
    of production planning.

Mathematical formulation
Formulate the Uncapacitated Lot Sizing Problem.
Define the decision variables.
Define the objective function minimizing total cost.
Write the constraints.
Clearly define all indices and variables.

Solution computation
Compute a feasible and optimized production plan based on the
   formulation.
Verify that:
all demands are satisfied
inventory flow is consistent across periods
setup decisions correctly correspond to production.

Solution method
- Clearly state the solution approach used (exact method,
   heuristic, or metaheuristic).
- If an exact method is used, describe the resolution process
   and stopping criteria.
- If a heuristic or local search is used, describe the algorithm
    (initialization, neighborhood moves, stopping criterion).
```

```
Solver-style output (MANDATORY)

Example format:
Optimal solution found
Objective value:
Enumerated nodes:
Total iterations:
Time (CPU seconds):
Time (Wallclock seconds):
Statut : Optimal
Cout total =

Tabular presentation of the solution

Present the final production plan in a clear table format with
   the following columns:
| Periode | Production | Stock | Setup |

Rules:
Each row must correspond to one planning period.
Ensure that inventory evolution is consistent between periods.
The table must contain exactly T rows.

Response time measurement
Explicitly report the time required to generate the complete
   answer (in seconds).
```

A.3.2 Prompt sans spécification de rôle

Listing A.6: Prompt du problème ULS sans spécification de rôle

```
2-Prompt : WITHOUT role specification

Solve the provided benchmark instance of the Uncapacitated Lot
   Sizing Problem (ULS) and explain the modeling and reasoning
   process.

Important constraints:
You are allowed to use the benchmark instance and its input data
   .
You are NOT allowed to directly recall, reproduce, or rely on
   previously published optimal solutions associated with this
   benchmark.
The solution must be obtained by explicitly performing the
   modeling and reasoning using only the provided data.

Instance format:
Line 1: Number of planning periods.
Line 2: Demand for each period.
```

```
Line 3: Inventory holding cost per unit carried from period t to
    t+1.
Line 4: Fixed setup cost incurred if production occurs in period
    t.
Line 5: Unit production cost.

Tasks:

Data understanding and parsing
Explain how the instance file is parsed.
Report the value of T.
Present the demand vector.
Present the holding cost vector.
Present the setup cost vector.
Present the unit production cost.
Briefly interpret what these parameters represent in the context
    of production planning.

Mathematical formulation
Formulate the Uncapacitated Lot Sizing Problem.
Define the decision variables.
Define the objective function minimizing total cost.
Write the constraints.
Clearly define all indices and variables.

Solution computation
Compute a feasible and optimized production plan based on the
   formulation.
Verify that:
all demands are satisfied
inventory flow is consistent across periods
setup decisions correctly correspond to production.

Solution method
- Clearly state the solution approach used (exact method,
   heuristic, or metaheuristic).
- If an exact method is used, describe the resolution process
   and stopping criteria.
- If a heuristic or local search is used, describe the algorithm
    (initialization, neighborhood moves, stopping criterion).

Solver-style output (MANDATORY)

Example format:
Optimal solution found
Objective value:
Enumerated nodes:
Total iterations:
Time (CPU seconds):
Time (Wallclock seconds):
Statut : Optimal
```

```
Cout total =

Tabular presentation of the solution

Present the final production plan in a clear table format with
   the following columns:
| Periode | Production | Stock | Setup |

Rules:
Each row must correspond to one planning period.
Ensure that inventory evolution is consistent between periods.
The table must contain exactly T rows.

Response time measurement
Explicitly report the time required to generate the complete
   answer (in seconds).
```

Annexe B

Instances de benchmark utilisées

Sources des benchmarks

Les instances utilisées dans ce travail proviennent de jeux de données de référence en optimisation combinatoire. Les liens des benchmarks sont indiqués ci-dessous afin de permettre la traçabilité des données expérimentales :

- ***p*-median** : instances OR-Library du problème p-median.
  https://people.brunel.ac.uk/~mastjjb/jeb/orlib/pmedinfo.html
- ***Set Covering Problem*** : instances OR-Library du problème de couverture par ensembles.
  https://people.brunel.ac.uk/~mastjjb/jeb/orlib/scpinfo.html
- ***Uncapacitated Lot Sizing Problem*** : instances ULS utilisées dans ce travail.
  https://github.com/lafilledepondy/ULS/tree/main/Instances_ULS

Les valeurs optimales de référence associées à ces instances sont utilisées pour comparer les solutions proposées par les LLM, notamment à travers le calcul du GAP.

B.1 Instances du problème *p*-median

Cette annexe présente les instances du problème p-median utilisées dans les expérimentations. Pour chaque instance, le tableau indique le nombre de nœuds n, le nombre d'arêtes m, le nombre de médianes à sélectionner p, ainsi que la valeur optimale de référence utilisée pour l'évaluation des solutions proposées par les LLM.

Table B.1: Instances p-median utilisées dans les expérimentations

Instance	n	m	p	**Solution optimale**
pmed1	100	200	5	5819
pmed4	100	200	20	3034
pmed7	200	800	10	5631
pmed10	200	800	67	1255
pmed13	300	1800	30	4374
pmed15	300	1800	100	1729
pmed17	400	3200	10	6999
pmed20	400	3200	133	1789
pmed23	500	5000	50	4619
pmed27	600	7200	10	8307
pmed30	600	7200	200	1989
pmed32	700	9800	10	9297
pmed35	800	12800	5	10400
pmed38	900	16200	5	11060
pmed40	900	16200	90	5128

B.2 Instances du *Set Covering Problem*

Cette section présente les instances du *Set Covering Problem* utilisées dans les expérimentations. Pour chaque instance, le tableau indique le nombre de lignes à couvrir m, le nombre de colonnes ou ensembles disponibles n, ainsi que la valeur optimale de référence utilisée pour évaluer les solutions proposées par les LLM.

Table B.2: Instances du *Set Covering Problem* utilisées dans les expérimentations

Instance	m	n	**Solution optimale**
scp41	200	1000	429
scp43	200	1000	516
scp45	200	1000	512
scp47	200	1000	430
scp49	200	1000	641
scp51	200	2000	253
scp53	200	2000	226
scp55	200	2000	211
scp57	200	2000	293
scp59	200	2000	279
scp61	200	1000	138
scp62	200	1000	146
scp63	200	1000	145
scp64	200	1000	131
scp65	200	1000	161

B.3 Instances du problème *Uncapacitated Lot Sizing*

Cette section présente les instances du problème *Uncapacitated Lot Sizing Problem* (ULS) utilisées dans les expérimentations. Pour les instances nommées sous la forme *T.numéro*, la première valeur correspond au nombre de périodes de planification T. La solution optimale indiquée correspond à la valeur de référence utilisée pour comparer les solutions proposées par les LLM.

Table B.3: Instances ULS utilisées dans les expérimentations

Instance	**Nombre de périodes T**	**Solution optimale**
toy	–	1687
21.1	21	16077
60.1	60	28801
60.2	60	25905
60.5	60	39667
90.1	90	52721
90.3	90	63105
90.5	90	73764
90.7	90	59714
90.9	90	64600
120.2	120	70975
120.4	120	94363
120.6	120	66894
120.8	120	79008
120.10	120	95775

Bibliographie

[1] D. Altit. Diabetes (1a parte). *Correo Farmacéutico*, 2003.

[2] J. E. Beasley. OR-Library: p-median test problems. https://people.brunel.ac.uk/~mastjjb/jeb/orlib/pmedinfo.html. Accessed: 2026-05-10.

[3] J. E. Beasley. OR-Library: Set covering problem test instances. https://people.brunel.ac.uk/~mastjjb/jeb/orlib/scpinfo.html. Accessed: 2026-05-10.

[4] Centrale Nantes. Laboratoire des sciences du numérique de nantes (LS2N). https://research.ec-nantes.fr/fr/laboratoires-de-recherche/laboratoire-des-sciences-du-numerique-de-nantes-ls2n, 2026. Consulté le 30 avril 2026.

[5] Yitian Chen, Jingfan Xia, Siyu Shao, Dongdong Ge, and Yinyu Ye. Solver-informed RL: Grounding large language models for authentic optimization modeling, 2025. arXiv:2505.11792.

[6] Ali Daud, Mobushira Khan, Sakher Ghanem, Sami Alesawi, Saud Yonbawi, Raed Alsini, Omar Alghushairy, Manal Linjawi, and Abdulrahman Ahmed Gharawi. Generative AI: A comprehensive overview of large language models for prompt engineering and applications. *arXiv preprint*, 2024. arXiv:2412.17690.

[7] Ashraf Elnashar, Jules White, and Douglas C. Schmidt. Enhancing structured data generation with GPT-4o: Evaluating prompt efficiency across prompt styles. *Frontiers in Computer Science*, 2024.

[8] Shengyu Feng, Weiwei Sun, Shanda Li, Ameet Talwalkar, and Yiming Yang. A comprehensive evaluation of contemporary ML-based solvers for combinatorial optimization, 2025. arXiv:2505.16952.

[9] Andrew Kean Gao. Prompt engineering for large language models: A brief guide with examples for non-technical readers. Stanford University, 2023. Unpublished manuscript.

[10] Gaïa. Uncapacitated Lot Sizing Problem benchmark instances. https://github.com/lafilledepondy/ULS/tree/main/Instances_ULS. Accessed: 2026-05-10.

[11] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Sebastopol, CA, 2 edition, 2019.

[12] Reza Ghanbarzadeh and Seyedali Mirjalili. A structured review of large language models in metaheuristic optimisation. *Decision Analytics Journal*, 15:100587, 2025.

[13] Andre Hottung, Federico Berto, Chuanbo Hua, Nayeli Gast Zepeda, Daniel Wetzel, Michael Römer, Haoran Ye, Davide Zago, Michael Poli, Stefano Massaroli, Jinkyoo Park, and Kevin Tierney. VRPAgent: LLM-driven discovery of heuristic operators for vehicle routing problems, 2025. arXiv:2510.07073.

[14] S. S. Hussain, N. Oliver, and D. C. Klonoff. *Insulin Pumps and Continuous Glucose Monitoring Made Easy E-Book*. Made Easy. Elsevier Health Sciences, 2015.

[15] IMT Atlantique. Création d'imt atlantique issue de la fusion de mines nantes et télécom bretagne. https://www.imt-atlantique.fr/fr/actualites/paul-friedel-directeur-d-imt-atlantique-est-elu-la-prestigieuse-academie-des 2019. Consulté le 30 avril 2026.

[16] IMT Atlantique. MODELIS : des solutions innovantes pour la logistique, la production et les services. https://www.imt-atlantique.fr/fr/actualites/modelis-logistique-innovantes, 2023. Consulté le 30 avril 2026.

[17] IMT Atlantique. Automation, production and computer sciences department (DAPI) – research. https://www.imt-atlantique.fr/en/about/departments/dapi/research, 2026. Consulté le 30 avril 2026.

[18] IMT Atlantique. Campus de nantes. https://www.imt-atlantique.fr/fr/campus/sites/nantes, 2026. Consulté le 30 avril 2026.

[19] IMT Atlantique. Ingénieur pour transformer l'avenir. https://www.imt-atlantique.fr/fr/l-ecole, 2026. Consulté le 30 avril 2026.

[20] LinkedIn. Comparing major AI disciplines: AI vs ML vs DL vs GenAI vs RAG vs AI agents. Publication LinkedIn, 2026. Consulté le 8 mars 2026.

[21] Sergio Morales, Robert Clarisó, and Jordi Cabot. Impromptu: A framework for model-driven prompt engineering. *arXiv preprint*, 2024. arXiv:2406.00213.

[22] M. Z. Naser. A review of prompt engineering techniques for large language models. *arXiv preprint*, 2023. arXiv:2407.12994.

[23] Krishna Ronanki, Simon Arvidsson, and Johan Axell. Prompt engineering guidelines for using large language models in requirements engineering. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2023.

[24] Nguyen Gia Hien Vu, Ke Wang, and G. Gary Wang. Effective prompting with ChatGPT for problem formulation in engineering optimization. *Engineering Applications of Artificial Intelligence*, 2024.

[25] Lilian Weng. Prompt engineering. Blog post, Lil'Log, 2023. Consulté le 10 avril 2026.

[26] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. A prompt pattern catalog to enhance prompt engineering with ChatGPT. *arXiv preprint*, 2023. arXiv:2302.11382.

[27] World Health Organization. Definition, diagnosis and classification of diabetes mellitus and its complications: report of a WHO consultation. Part 1, diagnosis and classification of diabetes mellitus, 1999. WHO/NCD/NCS/99.2.

[28] Bowen Zhang and Pengcheng Luo. OR-LLM-Agent: Automating modeling and solving of operations research optimization problem with reasoning large language model, 2025. arXiv:2503.10009.

[29] Qin Zhou and Jiuh-Biing Sheu. The use of generative artificial intelligence (GenAI) in operations research: review and future research agenda. *Journal of the Operational Research Society*, 2025.