INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES STRASBOURG

AUTODESK Make Anything

Développement de plugins Civil 3D assisté par IA

Rapport Projet de recherche technologique

Établissements : INSA / Autodesk

Date de rendu : 28 janvier 2026

Étudiant :
PELISSIER JEAN

Encadrants :
ALBY EMMANUEL
BERSON GUILLAUME
MACHER HÉLÈNE

TABLE DES MATIÈRES

0. Une assistance par intelligence artificielle a été utilisée ponctuellement pour la révision orthographique et l'amélioration stylistique de ce document.

REMERCIEMENTS

Je souhaite dans un premier temps remercier Autodesk de m'avoir donné l'opportunité de réaliser ce projet et d'avoir été encadré tout au long de cette expérience par Guillaume Berson. Il a toujours été disponible pour répondre à mes questions, m'a fourni tout le matériel nécessaire, et grâce à son expérience, j'ai pu mener à bien ce projet.

Aussi, je souhaite remercier mes encadrants de l'INSA, Mme.MACHER et M.ALBY, qui ont su être disponibles pour me guider dans mon travail et me fournir les outils de travail nécessaires au bon déroulement de notre étude.

Merci également à mes parents et à ma petite sœur pour leur soutien au cours des 5 dernières années d'études. Les 11 000 kilomètres qui nous séparent n'ont pas été suffisants pour nous séparer.

Un grand merci à mes amis pour tous les bons moments passés ensemble et tous ces rires échangés lors des revisions tardives.
Enfin, j'ai une pensée pour mes colocataires qui ont su me remotiver grâce à de bons plats quand le moral n'était pas au rendez-vous.

INTRODUCTION

L'intelligence artificielle occupe aujourd'hui une place croissante dans les outils numériques utilisés en ingénierie. Longtemps réduite à la réalisation de tâches d'analyse ou de calcul, l'IA est désormais utilisable directement dans les environnements de développement logiciel, en particulier à travers des assistants capables de générer, modifier et compiler du code. Cette évolution remet profondément en question la manière dont les ingénieurs conçoivent leurs outils, et plus largement la place du développement logiciel dans les métiers techniques.

Dans le domaine du génie civil et de la topographie, les logiciels de conception assistée par ordinateur jouent un rôle central dans les chaînes de production. Autodesk Civil 3D s'impose comme un logiciel de référence pour la conception d'infrastructures linéaires, et une partie de son potentiel repose sur la possibilité de développer des plugins adaptés aux besoins des utilisateurs. Ces extensions permettent d'automatiser des tâches répétitives, de fiabiliser les méthodes de travail et de gagner du temps sur certains projets. Pourtant, leur développement reste encore largement réservé à des profils spécialisés, maîtrisant à la fois le langage C# et l'API Autodesk.

Face à ce constat, l'émergence d'environnements de développement assistés par intelligence artificielle ouvre de nouvelles perspectives. Des outils comme Cursor promettent de réduire la barrière technique du développement en permettant à des utilisateurs non experts en programmation de concevoir leurs propres plugins, à condition de savoir dialoguer efficacement avec l'IA. La question ne devient alors plus uniquement technique, mais méthodologique : comment formuler correctement un besoin pour obtenir un résultat fiable ?

C'est dans ce contexte que s'inscrit ce projet de recherche technologique, réalisé en collaboration avec Autodesk. La problématique centrale est d'évaluer dans quelle mesure l'utilisation d'un assistant de développement IA peut faciliter la création de plugins Civil 3D, et comment structurer les interactions avec l'IA pour maximiser la qualité des résultats obtenus.

Pour répondre à cette problématique, un protocole expérimental avec une analyse des résultats de manière quantitative et qualitative a été mis en place, afin de dégager une méthode d'écriture de prompt optimisée et d'en discuter les apports, les limites et les perspectives.

Ce rapport présente successivement l'état de l'art, le protocole expérimental retenu, les résultats obtenus et leur analyse, avant d'ouvrir la discussion sur les enjeux plus larges liés à l'usage de l'intelligence artificielle dans le développement logiciel et l'évolution de nos métiers.

CHAPITRE

1

ÉTAT DE L'ART

1.1 Les enjeux du développement de plugins Civil 3D

Les logiciels de conception assistée par ordinateur occupent aujourd'hui une place centrale dans les métiers de l'ingénierie civile. Parmi eux, Autodesk Civil 3D est largement utilisé pour la conception et la gestion de projets d'infrastructures linéaires, tels que les routes, les réseaux ou les plateformes. Ce type de logiciel propose de nombreuses fonctionnalités standards, mais celles-ci ne couvrent pas toujours l'ensemble des besoins spécifiques rencontrés en pratique par les utilisateurs.

Pour répondre à ces besoins particuliers, Civil 3D permet le développement de plugins, c'est-à-dire des extensions logicielles venant s'ajouter au logiciel de base. Un plugin est un programme complémentaire qui peut automatiser une tâche, ajouter une nouvelle fonctionnalité ou adapter le comportement du logiciel à des paramètres spécifiques à l'entreprise. Contrairement aux fonctions natives, les plugins peuvent être conçus sur mesure afin de correspondre exactement aux méthodes de travail d'une entreprise, d'un service ou d'un projet.

L'utilisation de plugins constitue un levier important de productivité. En automatisant des opérations répétitives ou complexes, ils permettent de réduire le temps passé sur des tâches manuelles, tout en limitant les erreurs humaines. Cette automatisation favorise également une meilleure standardisation des méthodes, en garantissant que les mêmes règles de calcul, de dessin ou de vérification sont appliquées d'un projet à l'autre. Ces aspects sont mis en avant dans les travaux de (RAMPF, 2017) portant sur l'optimisation de la génération de plugins.

Au-delà du gain de temps, les plugins contribuent à améliorer la fiabilité des productions. En paramétrant une logique métier précise dans un outil logiciel, ils réduisent la dépendance des livrables à l'interprétation individuelle de l'utilisateur. Le plugin permet de retracer les étapes de calcul et de conception, assurant ainsi une transparence sur les résultats et les choix effectués. Dans un contexte professionnel, cela représente un enjeu majeur, notamment pour la reproductibilité des résultats et le respect des normes de chantier.

Cependant, malgré leurs avantages, les plugins Civil 3D restent encore peu développés en dehors de cercles spécialisés. Leur conception nécessite des compétences en programmation, ce qui les rend inaccessibles à de nombreux ingénieurs ou techniciens. Cette difficulté explique pourquoi, dans de nombreuses structures, les chaînes de traitement restent entièrement manuelles, même si la possibilité d'automatisation existe.

Ainsi, les plugins apparaissent comme des outils à fort potentiel, mais dont l'exploitation reste li-

mitée par la complexité de leur développement. Cet écart entre l'intérêt métier élevé et la barrière technique importante constitue un point de départ pour comprendre l'intérêt d'approches reposant sur l'assistance par intelligence artificielle.

1.2 Le développement classique de plugins Civil 3D

Le développement de plugins pour Civil 3D repose sur l'accès aux fonctionnalités internes du logiciel via son API (*Application Programming Interface*), qui fournit les fonctions nécessaires pour automatiser des tâches courantes : alignements, surfaces, profils, blocs ou calques. Autodesk fournit des guides et des références pour exploiter cette API (AUTODESK, 2025a), (AUTODESK, 2025b).

Civil 3D s'appuie sur l'environnement .NET et le langage C# pour le développement de plugins. Les plugins sont chargés dans Civil 3D sous la forme d'une DLL, et le code C# appelle les fonctions de l'API fournies dans les DLL Autodesk. Le processus type, détaillé par Rampf, suit ces étapes : configurer le projet C# avec les références Autodesk, implémenter la logique métier via l'API, compiler le code C# pour obtenir la DLL puis la charger dans Civil 3D via la commande *NETLOAD* (RAMPF, 2017).

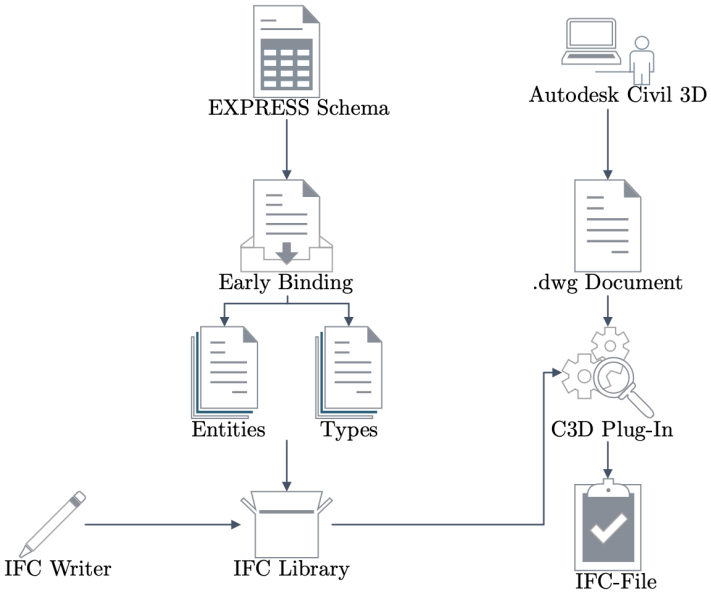


FIGURE 1.1 – Étapes du développement d'un plugin pour la création d'axes IFC selon (RAMPF, 2017).

Ce fonctionnement offre une large personnalisation mais exige une compréhension approfondie du langage C#, de l'API Civil 3D et de l'environnement .NET. Cela limite l'accès à des profils spécialisés, ce qui est rarement le cas des ingénieurs en génie civil.

1.3 L'IA comme réponse aux limites du développement classique

Les difficultés d'apprentissage et le risque d'erreurs ont favorisé l'usage d'intelligences artificielles génératives. Les *Large Language Models* (LLM) sont des IA qui apprennent sur du code, de la documentation et des échanges de communautés de développeurs. Ils découpent le texte en unités appelées *tokens* et évaluent, à chaque étape, la probabilité du token suivant en fonction du contexte (PARTHASARATHY et al., 2024).

Appliqués au code, ces modèles peuvent proposer des fonctions complètes, générer des environnements de code ou aider à corriger des erreurs. La communication avec ces outils se fait en langage naturel, ce qui réduit la barrière d'entrée et permet de déléguer à l'IA des tâches répétitives. Ce sont justement les tâches jugées les moins attractives que les développeurs souhaitent automatiser selon des études récentes (SERGEYUKA et al., 2024).

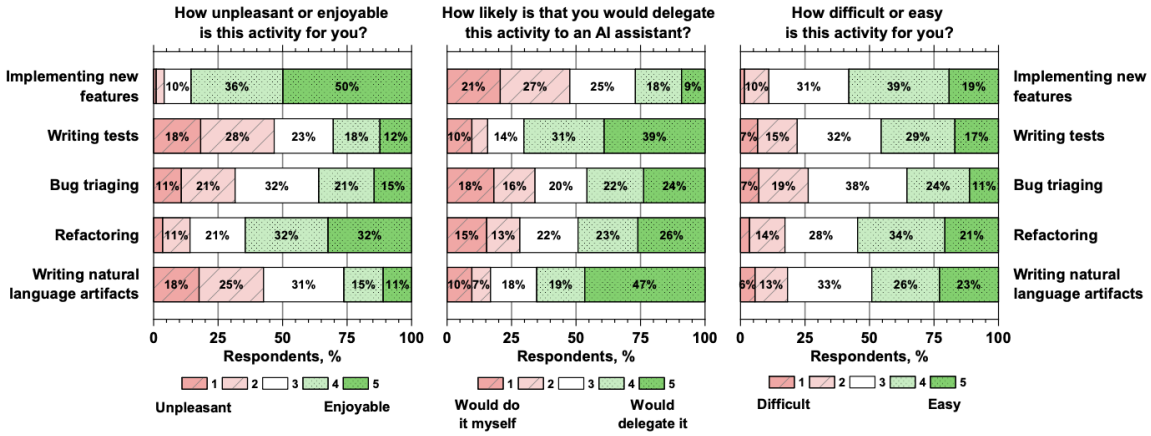


FIGURE 1.2 – Comparaison entre l'attrait des tâches techniques et la volonté des développeurs de les déléguer à une IA (Source : (SERGEYUKA et al., 2024))

Les travaux de (THAW, 2025) montrent une hausse de l'utilisation de l'IA de la part des développeurs. Il en ressort aussi une augmentation de la productivité de ces développeurs grâce à l'IA, à condition de conserver une supervision humaine. À l'inverse, (BECKER et al., 2025) observent que l'assistance dégrade la performance de développeurs expérimentés.

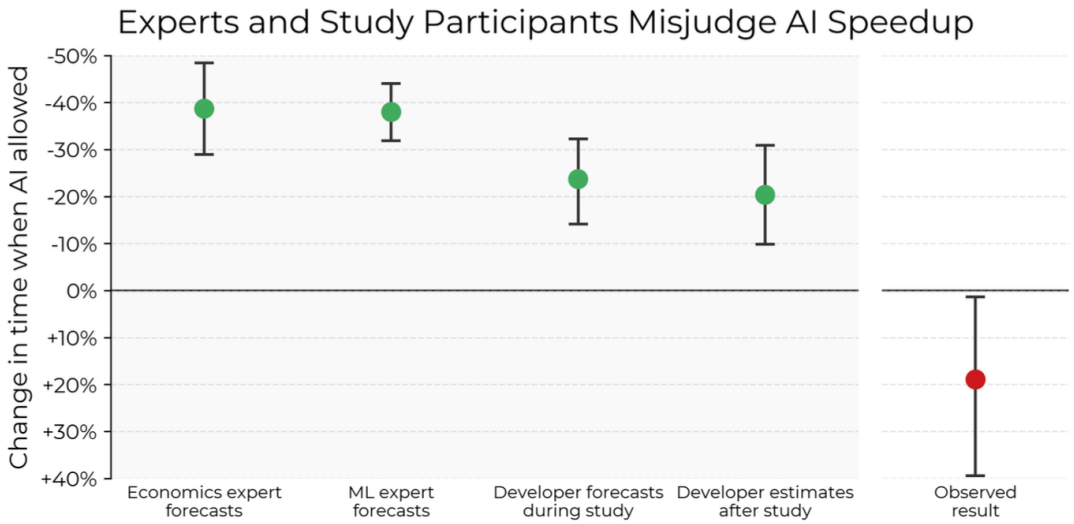


FIGURE 1.3 – Écart entre la productivité perçue et la réalité. (Source : (BECKER et al., 2025))

Cette étude cible donc des développeurs novices ou intermédiaires pour qui l'utilisation de l'IA est indispensable.

La limite principale de ces modèles est qu'ils restent dépourvus de compréhension des besoins métier. En cas d'instructions floues, l'IA complète donc le contexte par des hypothèses plausibles mais

parfois incorrectes. La qualité du code dépend donc entièrement de la précision et de la structure des consignes (THAW, 2025).

Enfin, les modèles conversationnels comme ChatGPT ou Gemini ne permettent pas de modifier, compiler ou structurer du code. En effet, ils sont efficaces pour expliquer ou donner des exemples, mais leur action reste indirecte tant qu'ils n'ont pas accès aux fichiers, à l'arborescence et aux outils de compilation du projet.

D'où l'émergence d'environnements de développement intégrés augmentés par l'IA (*AI-powered IDE*). Un IDE regroupe édition, gestion des fichiers, compilation et débogage. L'ajout d'un modèle de langage permet une interaction continue entre le développeur, le code et l'IA, ce qui facilite l'appropriation de projets complexes, surtout pour des non-experts.

L'IDE Cursor s'inscrit dans cette dynamique : il regroupe les outils d'un IDE classique et intègre des LLM (ChatGPT, Claude) pour accompagner l'utilisateur. Dans cette logique de *Vibe Coding*, s'il est bien guidé, Cursor peut préparer l'environnement nécessaire au plugin Civil 3D, structurer les dossiers du projet, piloter l'écriture et la compilation du code C# (RAY, 2025).

D'autres solutions existent comme Visual Studio Code avec GitHub Copilot, l'IDE Rider avec l'assistant JetBrains AI, ou VS Code accompagné de Codex. La littérature rappelle toutefois certains problèmes liés à ces outils : la génération de fonctions inexistantes, la confusion entre différentes versions d'API, ou la génération de code syntaxiquement correct mais inadapté ou trop lourd (PARTHASARATHY et al., 2024 ; HAJIPOUR, 2024).

Ces travaux s'accordent donc pour considérer l'IA comme un assistant de développement plutôt qu'un substitut au raisonnement humain. La connaissance métier et la précision des instructions restent déterminantes et constituent l'axe central de notre projet.

CHAPITRE

2

PROTOCOLE

L'état de l'art a mis en évidence deux constats principaux : d'une part, le fort potentiel des plugins Civil 3D pour améliorer la productivité et la fiabilité des workflows, et d'autre part, les limites du développement classique liées à la complexité du langage C# et de l'API. Les travaux récents montrent également que l'intelligence artificielle peut constituer une réponse pertinente à ces difficultés, à condition d'être correctement utilisée et encadrée.

Dans ce contexte, l'objectif de ce projet n'est pas uniquement d'évaluer la capacité d'une IA à générer du code, mais de comprendre dans quelles conditions cette assistance devient réellement efficace. La qualité des résultats fournis par une IA générative dépend en grande partie de la manière dont les instructions lui sont formulées, c'est-à-dire du prompt.
Il apparaît donc nécessaire de structurer une méthodologie permettant d'analyser l'influence du niveau de détail, de précision et de structuration d'un prompt sur le développement de plugins Civil 3D.

La méthodologie présentée dans ce chapitre décrit l'approche retenue pour étudier cet apport. Elle repose sur une organisation progressive du projet, la définition d'un protocole expérimental basé sur plusieurs niveaux de prompts, et la mise en place d'indicateurs permettant d'évaluer de manière objective les résultats obtenus.

Ce chapitre détaille successivement la structure générale du projet, les outils choisis, la construction du protocole de test centré sur la rédaction de prompts, ainsi que les critères utilisés pour analyser les performances et les limites de l'assistance par intelligence artificielle.

2.1 Approche générale et outils

Afin d'évaluer l'apport de l'intelligence artificielle dans le développement de plugins Civil 3D, il a d'abord été nécessaire de mettre en place un environnement de travail fonctionnel, permettant à la fois le développement classique en C# et l'assistance par un IDE augmenté par l'IA. La compilation en .NET pour Civil 3D étant exclusivement disponible sous Windows, l'ensemble du projet a dû être réalisé sur ce système d'exploitation, ce qui exclut toute possibilité de développement natif sous MacOS. Cette contrainte, bien que classique dans le contexte Autodesk, doit être explicitée car elle influence le choix dans les versions des outils utilisés.

Le premier élément indispensable est naturellement l'installation de Civil 3D lui-même, qui fournit non seulement l'environnement cible du plugin, mais également les bibliothèques nécessaires à son développement. Ces bibliothèques prennent la forme de fichiers DLL Autodesk, comme présenté dans

l'état de l'art, qui exposent les fonctionnalités internes du logiciel via son API. On peut distinguer deux bibliothèques distinctes. D'un côté, les DLL liées à AutoCAD telles que *acdbmgd.dll* ou *acmgd.dll*, qui regroupent les fonctions de base de dessin et de gestion des entités. De l'autre côté on retrouve les bibliothèques spécifiques à Civil 3D comme *AeccDbMgd.dll* ou *AecBaseMgd.dll*, qui donnent accès aux objets métiers propres aux infrastructures comme les axes, profils, surfaces, réseaux. Dans la pratique, l'absence ou une mauvaise référence des chemins menant à ces DLL conduit à des erreurs de compilation ou d'exécution, un point qui s'est révélé central lors des premiers tests.

Un second élément fondamental de l'environnement est le .NET SDK. L'installation de Civil 3D seule ne suffit pas à permettre la compilation d'un plugin. Le SDK .NET fournit le compilateur C#, l'outil MSBuild et l'ensemble de la chaîne de génération des bibliothèques dynamiques que l'on chargera via la commande *NETLOAD* dans le logiciel. Sans ce SDK, il est possible d'écrire du code source, mais impossible de produire le plugin exploitable par Civil 3D (MICROSOFT, 2025). Lors des premières tentatives de compilation assistées par l'IA, Cursor était capable de générer un code syntaxiquement correct, mais échouait systématiquement à produire une DLL tant que le SDK n'était pas installé et correctement configuré.

L'environnement de développement s'appuie ensuite sur l'IDE Cursor, présenté dans l'état de l'art comme un IDE augmenté par l'IA. Dans le cadre de ce projet, Cursor a été utilisé comme outil principal de développement, en remplacement d'un IDE classique comme VSCode ou PyCharm. Plusieurs versions de Cursor existent, avec des capacités variables selon les modèles de langage utilisés. L'entièreté de ce projet a été réalisée avec la version 1.9.0 de Cursor Pro et le LLM de Openai GP-Codex-5. L'interface de l'IDE Cursor a été spécifiquement conçue pour intégrer nativement les capacités des modèles de langage au cœur de l'environnement de développement. La figure 2.1 illustre cette intégration en présentant la disposition de l'espace de travail utilisé durant notre protocole.

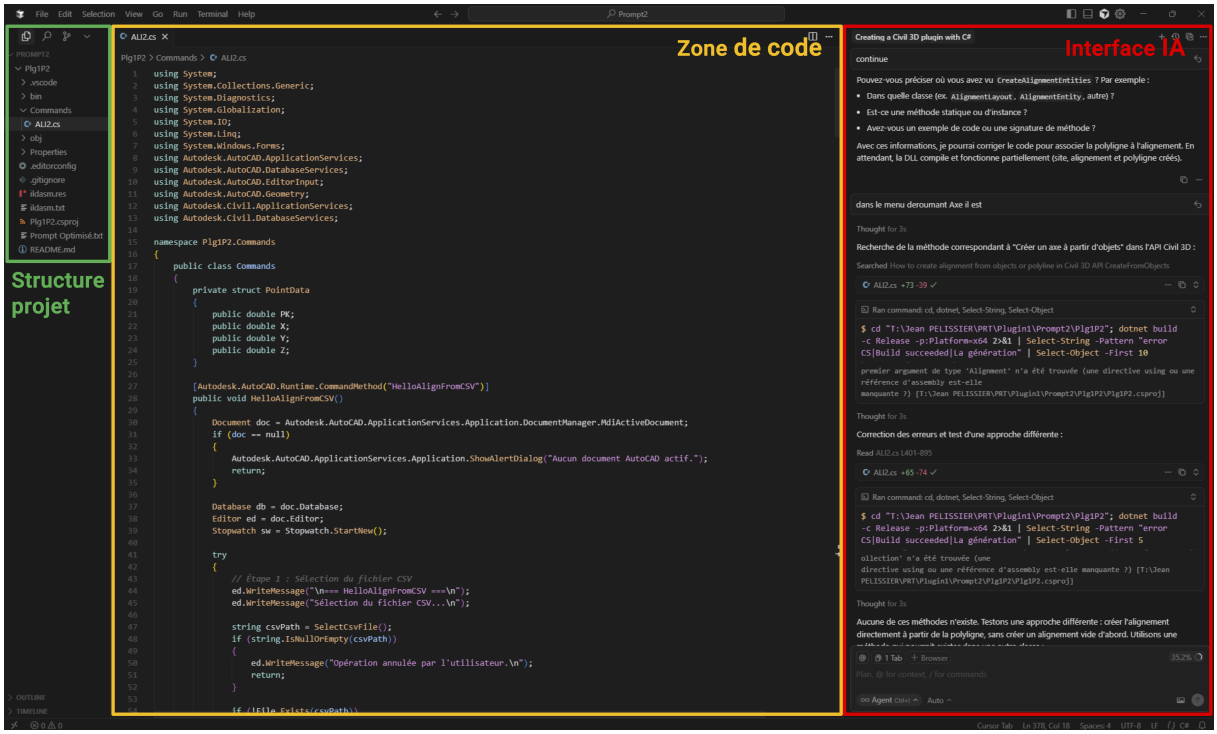


FIGURE 2.1 – Interface de l'IDE Cursor

Cette organisation en trois zones distinctes, visible sur la capture d'écran ci-dessus, permet de maintenir un flux de travail continu sans rupture de contexte :

— **À gauche (Structure du projet) :** L'arborescence classique des fichiers permettant une navi-

gation rapide entre les classes C# et les fichiers de configuration `.csproj`.
— **Au centre (Zone de code) :** L'éditeur de texte principal où le code généré ou modifié par l'IA peut être vérifié, testé et édité manuellement.
— **À droite (Interface IA) :** Le panneau de discussion (Chat) où s'effectue la conversation avec le modèle de langage, permettant de formuler les prompts et de recevoir les suggestions de correction en temps réel.

Pour les expérimentations, Cursor a été configuré en mode automatique, lui permettant d'exécuter les différentes étapes de développement (création de fichiers, modification du code, compilation) sans validation manuelle systématique (CURSOR, 2025). Ce choix a pour but de rapprocher l'expérimentation d'un usage réaliste en contexte professionnel, tout en conservant un contrôle a posteriori sur les résultats produits.

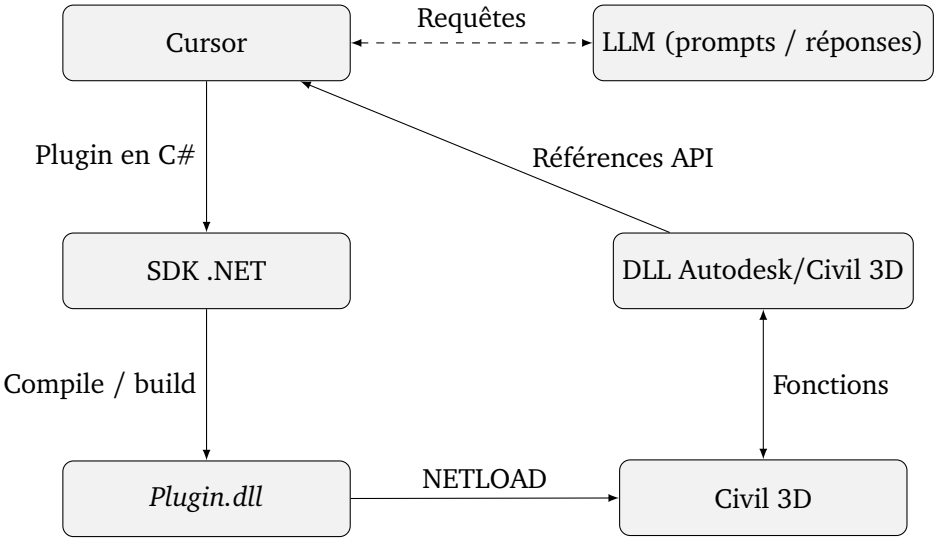


FIGURE 2.2 – Schéma des connexions entre les différents éléments de l'environnement de développement de plugin Civil 3D assisté par IA.

La mise en place de cet environnement n'a pas été immédiate et a nécessité plusieurs ajustements. Certaines erreurs récurrentes, telles que des références croisées entre DLL AutoCAD et Civil 3D ou des incohérences de version .NET, ont mis en évidence l'importance d'un environnement correctement installé avant toute expérimentation sur les prompts. Ces difficultés initiales ont permis d'identifier les prérequis techniques indispensables à un développement assisté par l'IA réellement opérationnel, et ont guidé la structuration méthodologique du projet.

Ainsi, cette phase de mise en place ne constitue pas une simple étape préparatoire, mais une base essentielle sur laquelle repose l'ensemble de notre étude. Elle permet de garantir que les résultats observés par la suite sont bien liés à l'usage de l'intelligence artificielle et à la qualité des prompts, et non à des défauts de configuration de l'environnement de développement.

2.2 Construction du protocole expérimental

Suite à la mise en place de l'environnement de développement, l'étape suivante a consisté à définir un protocole expérimental permettant de répondre à notre problématique.

2.2.1 Identification du prompt comme variable centrale

Un premier test a été réalisé afin de répondre à une question fondamentale : l'IA est-elle capable de développer un plugin Civil 3D fonctionnel ? Ce test consistait à demander à Cursor de créer un plugin extrêmement simple permettant de générer une polyligne 2D entre deux coordonnées saisies par l'utilisateur. Bien que ce plugin mobilise uniquement des fonctions basiques d'AutoCAD, il permet de vérifier la capacité de l'IA à interagir avec l'API, gérer les entrées utilisateur et produire un code compilable.

Après plusieurs itérations et retours d'erreurs, Cursor a finalement été en mesure de produire un plugin fonctionnel. La faisabilité du développement assisté par IA a donc été confirmée. Cependant, ce premier essai a mis en évidence que notre processus était peu maîtrisé, marqué par de nombreuses erreurs de compilation et des références manquantes.

En effet, Cursor retourne explicitement des messages d'erreur indiquant des éléments absents du prompt comme les chemins vers les DLL Autodesk, la version du logiciel, la version du SDK .NET ou encore la structure attendue des fichiers. Il en resort que ces difficultés n'étaient pas liées à la complexité du plugin, mais à l'imprécision des prompts fournis à l'IA. Cette observation conduit à identifier le prompt comme la variable centrale de notre étude.

2.2.2 Prompt de création de l'environnement de développement

Avant toute description de la logique d'un plugin, il est indispensable de fournir à l'IA un prompt détaillant précisément l'environnement de travail. Cette partie du prompt consacrée à l'initialisation a été systématiquement utilisée pour l'ensemble des tests et constitue un prérequis indépendant au plugin développé.

Cette partie du prompt doit contenir :

- Le système d'exploitation utilisé (ici Windows 11), c'est une information indispensable au fonctionnement de Civil 3D et du .NET SDK.
- La version d'AutoCAD et de Civil 3D, afin d'éviter toute ambiguïté liée aux évolutions de l'API.
- La version du .NET SDK et l'utilisation explicite de la ligne de commande *dotnet build* pour la compilation.
- Toute la structure du projet et le chemin vers le dossier racine du projet.
- Les chemins précis vers les DLL AutoCAD et Civil 3D (*acdbmgd.dll, acmgd.dll, AeccDbMgd.dll*, etc.).
- Les paramètres de compilation (cible net8.0-windows, plateforme x64, génération d'une DLL en configuration Release).

Ce prompt impose également une structure de projet standardisée : création d'un projet SDK.NET-style, organisation des fichiers source, la présence d'un fichier `.csproj` correctement référencé et automatisation complète du processus de build. L'objectif est d'obtenir une DLL exploitable dans Civil 3D sans intervention manuelle.

Vous retrouverez ce prompt pour la création de l'environnement dans l'annexe (Annexe A.1).

2.2.3 Définition des niveaux de prompt pour la logique du plugin

Une fois l'environnement correctement défini, il faut se concentrer sur la description fonctionnelle du plugin à travers différents niveaux de prompt. Trois niveaux ont été définis : P0, P1 et P2. Ils correspondent à des degrés croissants de structuration et de précision, et ont été conçus pour représenter des profils d'utilisateurs différents.

Prompt P0 – Prompt minimaliste Le prompt P0 correspond à une formulation primaire du fonctionnement. Ce prompt est proche de ce qu'un utilisateur débutant ou pressé pourrait écrire. On se

rapproche d'une demande qu'on pourrait faire à l'oral. Il se limite à une description générale de la fonctionnalité attendue, sans préciser ni les étapes, ni les contraintes techniques.

Par exemple, pour notre premier plugin test, P0 serait :

> « Créer un plugin Civil 3D qui trace une polyligne entre deux points. »

Ce type de prompt laisse à l'IA une grande liberté d'interprétation. Il ne précise ni le type de commande attendue, ni la gestion des entrées utilisateur, ni les classes de l'API à utiliser. P0 servira de référence pour évaluer le comportement de l'IA en situation de sous-spécification.

Prompt P1 – Prompt structuré basique Le prompt P1 correspond à une description plus détaillée qu'un utilisateur motivé, mais non expert, pourrait formuler. Il donne une structure logique et des indications fonctionnelles mais sans imposer une méthodologie complète.

Par exemple, pour le plugin de polyligne, P1 serait :

> « Créer un plugin Civil 3D avec une commande qui demande deux points à l'utilisateur et trace une polyligne 2D entre ces points. La commande doit afficher des messages clairs et gérer les erreurs. »

Ce niveau apporte déjà des informations sur les entrées utilisateur et le comportement attendu, mais laisse encore à l'IA le soin de choisir l'architecture du code et les classes de l'API. P1 représente un prompt qu'un ingénieur ferait naturellement, fréquemment observé lors d'une utilisation spontanée des outils d'IA (CHEN et al., 2025).

Prompt P2 – Prompt structuré optimisé Le prompt P2 correspond au niveau le plus abouti de rédaction testé dans ce projet. Il repose sur une structuration explicite du prompt, construite à partir des erreurs observées lors des premiers tests mais aussi des travaux de (CHEN et al., 2025 ; HAJIPOUR, 2024 ; PARTHASARATHY et al., 2024).

Contrairement aux niveaux précédents, le prompt P2 ne se limite pas à une description fonctionnelle du plugin. La méthode d'écriture vise à encadrer le raisonnement de l'IA et à réduire les interprétations implicites.

La structure du prompt P2 s'articule systématiquement autour des éléments suivants :

- Un contexte de développement clairement défini (Besoin métier, cadre du projet, limites techniques).
- Un objectif fonctionnel explicite.
- Les entrées et sorties attendues.
- Une description séquentielle des étapes de traitement (logique du plugin).
- Des contraintes techniques.
- Des critères de validation du résultat.

Les étapes de traitement sont précisées dans un ordre explicite, afin de contraindre l'enchaînement logique du code généré.

Enfin, le prompt P2 intègre explicitement :

- Des contraintes de compatibilité et de performance.
- Des règles de gestion des erreurs.
- Des critères permettant d'évaluer si le résultat produit est conforme au comportement attendu.

Ce niveau de structuration vise à réduire au maximum l'ambiguïté du prompt et à canaliser le comportement probabiliste du LLM.

Une fois la logique du plugin correctement définie, notamment pour P2, le prompt peut être rédigé ou affiné avec d'autres IA conversationnelles telles que ChatGPT ou Gemini. Ces outils permettent de clarifier le besoin et de structurer le prompt, avant son utilisation dans Cursor.

Vous retrouverez un exemple complet du prompt P2 pour le plugin de polyligne en annexes (Annexe A.2).

2.2.4 Choix des plugins testés

Afin d'évaluer ces niveaux de prompt sur des cas concrets, trois plugins ont été définis. Ils ont été choisis pour couvrir un maximum de fonctionnalités et l'utilisation de différentes parties de l'API AutoCAD et Civil 3D, avec une complexité croissante.

Le premier plugin correspond à la création d'un axe à partir d'un fichier CSV contenant des coordonnées. Ce plugin mobilise des fonctions de lecture de fichiers, de création d'entités Civil 3D (axes) et de gestion des transactions.

Le second plugin dans la continuité du premier permet de poser des blocs le long d'un axe à des intervalles définis. Ici on pourra définir un sens de pose, un espacement, et le type de bloc à insérer. Ce plugin nécessite une interaction plus poussée avec l'API Civil 3D, notamment pour manipuler les axes et insérer des blocs.

Le troisième plugin implique l'analyse de fichiers *.dwg* dans un dossier et l'extraction d'informations de ces fichiers. L'idée est de sortir un rapport listant toutes les entités spécifiques que contiennent les fichiers DWG d'un dossier sélectionné. Ce plugin est plus complexe car il combine la gestion de fichiers, l'analyse de dessins et la génération de rapports.

Le choix de ces plugins a été effectué à la suite d'échanges avec Guillaume Berson, qui a identifié les fonctionnalités souvent demandées par les utilisateurs sur des blogs et forums de la communauté Civil 3D (BERSON, 2025).

2.2.5 Cadre expérimental

Le développement de chaque plugin a été tenté avec chacun des trois niveaux de prompt (P0, P1 et P2), ce qui conduit à un total de neuf tests distincts. Pour chaque test, on a créé un dossier vierge afin qu'il n'y ait aucune dépendance entre les tests. Aussi, on fournira à Cursor les messages d'erreurs retournés lors du test s'il y en a, afin de lui permettre de corriger son code. Les erreurs pourront être transmises telles quelles à l'IA, soit reformulées lorsque l'origine du problème est comprise.

Afin d'encadrer l'expérimentation, un critère d'arrêt a été défini. Un test est considéré comme concluant lorsque le plugin compile correctement et fonctionne dans Civil 3D dans un temps maximal de vingt minutes. À l'inverse, le test est interrompu si l'IA entre dans une boucle d'erreurs répétitives sans amélioration notable ou qu'il dépasse le temps imparti de 20 minutes.

Algorithm 1 Algorithme du protocole expérimental

```
prompts ← [P0, P1, P2] ; plugins ← [1, 2, 3]
for all plg ∈ plugins do
    for all P ∈ prompts do
        Créer un dossier de test vierge.
        Démarrer un chronomètre.
        while plugin non opérationnel ou pas d'erreurs répétitives ou chrono < 20 minutes do
            Lancer la compilation dans Cursor pour obtenir le plugin.dll.
            Charger l'extension dans Civil 3D via NETLOAD.
            Envoyer les messages d'erreurs à l'IA (bruts ou reformulés si l'origine est comprise).
        end while
        if plugin opérationnel then
            Marquer le test « concluant »
        else
            Marquer le test « non concluant».
        end if
    end for
end for
```

Ce cadre permet d'assurer une comparaison équitable entre les différents niveaux de prompt et de préparer l'analyse des performances et des limites de l'IA, présentée dans la section suivante.

2.3 Outils d'analyse

Afin d'évaluer l'influence du niveau de prompt sur la capacité de l'IA à développer un plugin Civil 3D fonctionnel, il a été nécessaire de définir un ensemble d'indicateurs permettant de caractériser objectivement et subjectivement chaque expérimentation. Cette section se concentrera sur les paramètres mesurés et la manière dont les résultats ont été notés.

L'objectif principal de ces mesures est d'en tirer un prompt optimal selon la complexité du plugin, et de comprendre dans quelles conditions l'IA produit des plugins fiables, stables et exploitables. Les indicateurs retenus portent soit sur la performance temporelle du développement, la qualité du code généré ou l'expérience du développement et d'utilisation.

2.3.1 Mesures temporelles

Deux mesures temporelles distinctes ont été notées lors de chaque test :

- T_1 : le temps écoulé entre le lancement du prompt et l'obtention du premier *build* réussi, même si le plugin n'était pas encore pleinement fonctionnel.
- T_2 : le temps nécessaire pour obtenir un plugin fonctionnel conforme au comportement attendu.

Les phases de test manuel du plugin dans Civil 3D ont volontairement été exclues de ces mesures. Le temps mesuré correspond uniquement au temps de développement assisté par l'IA, afin de se concentrer sur la performance propre de l'outil et du prompt, indépendamment de l'intervention humaine.

Les temps ont été mesurés à l'aide d'un chronomètre à la main dont il est possible d'estimer la précision des mesures à ± 1 seconde. Cette précision est jugée suffisante au regard des ordres de grandeur observés, les phases de développement s'étalant généralement sur plusieurs minutes.

Indicateurs de développement

Pour chaque test, le nombre de tentatives de compilation a été noté. Cet indicateur permet d'évaluer la capacité de l'IA à converger vers une solution compilable et sa capacité à seule ses erreurs.

La taille finale du dossier permettant de générer le plugin a également été relevée. Bien que ce paramètre ne soit pas un indicateur de qualité, il permet d'identifier si l'IA a généré trop ou pas assez de fichiers source, et d'évaluer la complexité du code produit.

2.3.2 Évaluation fonctionnelle du plugin

Chaque plugin généré a été évalué selon huit critères fonctionnels, notés individuellement. Ces critères notés sur 1 point chacun pour avoir une note finale sur 8 points sont les suivants :

- La conformité du plugin par rapport à l'objectif initial.
- La robustesse face aux erreurs utilisateur (saisie invalide, annulation).
- La stabilité globale (absence de crash ou de comportement inattendu).
- Le bon chargement de la DLL dans Civil 3D.
- L'exécution correcte des commandes attendues.
- La précision des résultats produits par le plugin comparée aux résultats trouvés sans utiliser le plugin.
- La qualité de la gestion des erreurs dans la ligne de commande.
- La clarté des messages et instructions fournis à l'utilisateur.

Ces critères permettent d'évaluer non seulement le fonctionnement du plugin, mais également sa facilité d'utilisation dans un contexte réel.

2.3.3 Retours qualitatifs

En complément des indicateurs quantitatifs, deux types de commentaires qualitatifs ont été systématiquement rédigés :

- Un retour en tant que développeur, décrivant le déroulement du développement (fluidité, présence de boucles d'erreurs, corrections successives, comportement de l'IA).
- Un retour en tant qu'utilisateur, évaluant l'ergonomie, la clarté et l'intuitivité du plugin.

Ces retours sont forcément un peu subjectifs mais ils permettent une trace du déroulement des expérimentations.

2.3.4 Auto-évaluation du prompt

Aussi, lors de chaque test, il a été ajouté dans le prompt une partie demandant à Cursor de fournir un retour critique sur le prompt utilisé. Cursor devait identifier les éléments superflus, les informations manquantes et proposer une version optimisée du prompt.

Cette démarche permet de confronter l'analyse humaine à l'auto-évaluation du modèle, et de trouver des pistes d'amélioration pour la rédaction de prompts intermédiaires entre les niveaux P1 et P2.

2.3.5 Synthèse des résultats

L'ensemble des paramètres mesurés, des commentaires qualitatifs et des prompts utilisés a été consigné dans un fichier Excel récapitulatif. Cette organisation permet de garantir la traçabilité des tests, de reproduire les expérimentations dans des conditions identiques et de faciliter l'analyse comparative présentée dans le chapitre suivant.

Bien que certains indicateurs reposent sur une appréciation subjective, cette approche mixte permet d'obtenir une vision globale du comportement de l'IA face à différents niveaux de structuration du prompt, sans présumer à ce stade des conclusions qui en seront tirées.

CHAPITRE

3

RÉSULTATS ET ANALYSES

Le chapitre précédent a permis de définir le protocole expérimental ainsi que les indicateurs retenus pour évaluer le développement de plugins Civil 3D assisté par l'IA, mais aussi l'impact du prompt sur ce développement.

Ce chapitre présente et analyse les résultats obtenus lors des différentes expérimentations. L'objectif est d'évaluer, de manière comparative, l'influence du niveau de structuration du prompt sur la performance de développement, la qualité des plugins générés et le comportement global de l'IA. Cette partie ne sera pas une présentation des résultats bruts, mais une synthèse mettant en lumière les tendances majeures et les observations tirées des données collectées.

3.1 Présentation des résultats

L'ensemble du protocole expérimental décrit au chapitre précédent a pu être mené. Au total, neuf tests ont été réalisés, correspondant à la combinaison des trois niveaux de prompt (P0, P1 et P2) avec les trois plugins définis.

Parmi ces neuf tests, six ont abouti à des plugins parfaitement fonctionnels. Un test est considéré comme abouti lorsque le plugin compilé se charge correctement dans Civil 3D et qu'il est conforme au besoin initialement défini.

Trois tests n'ont pas permis d'obtenir un plugin satisfaisant. Dans deux cas, correspondant au premier plugin avec les prompts P0 et P1, l'IA est entrée dans des boucles d'erreurs successives. Le problème rencontré venait d'une mauvaise décision de l'IA pour la création d'un axe : elle passait par le dessin d'une polyligne et n'arrivait pas à la transformer en axe. Malgré plusieurs itérations et retours d'erreurs fournis, l'IA changeait constamment la même chose et n'arrivait pas à corriger le problème. Dans un troisième cas, correspondant au troisième plugin avec le prompt P1, le code généré n'a pas permis de produire une DLL correctement chargeable dans Civil 3D malgré les explications fournies à Cursor.

Ces tests non aboutis n'ont donc pas été interrompus par la limite de temps de 20 minutes mais lorsque l'IA ne parvenait plus à progresser vers une solution fonctionnelle. Les résultats sont néanmoins exploitables du point de vue expérimental, car tous les paramètres cités dans la partie précédente ont pu être mesurés.

L'ensemble des résultats bruts, incluant les temps mesurés, le nombre de tentatives de compilation,

les commentaires qualitatifs et les prompts utilisés, sont présentés en annexe (voir Annexe A.3).

3.2 Analyse des résultats

3.2.1 Analyse quantitative des résultats

Afin d'analyser objectivement l'impact des différents niveaux de prompt sur le processus de développement assisté par l'IA, une première analyse quantitative a été menée à partir des temps de développement mesurés pour chaque test T1 et T2.

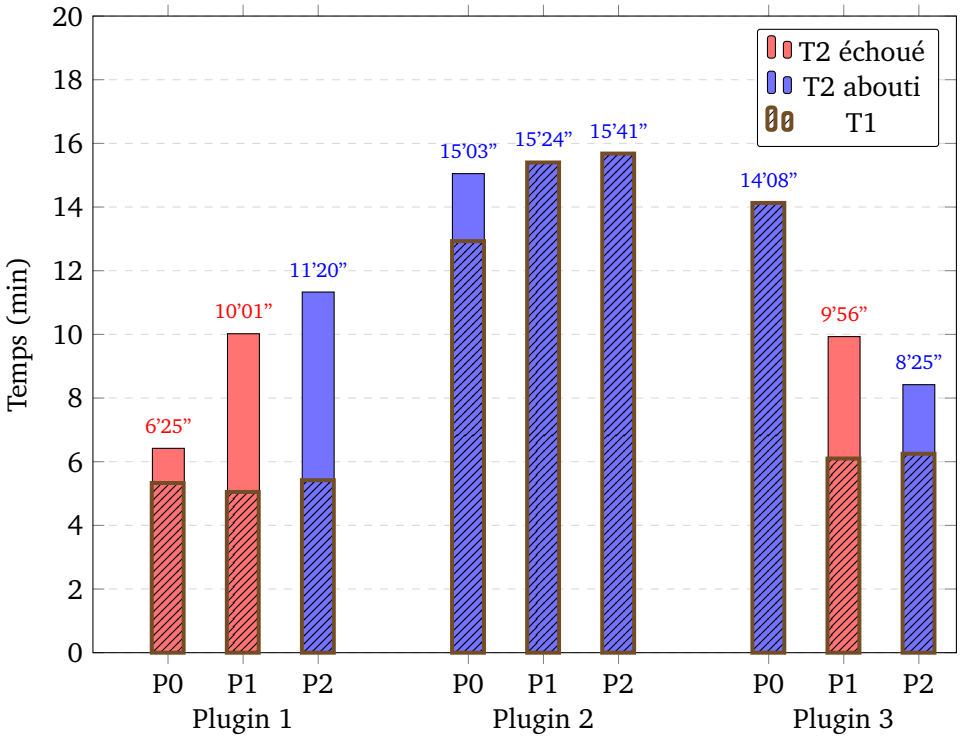


FIGURE 3.1 – Représentation de T1 et T2 selon les niveaux de prompt et de plugin

Cette figure présente, pour chacun des plugins et pour chaque niveau de prompt, le temps jusqu'au premier build T1 (hachuré) ainsi que le temps jusqu'à l'obtention d'un plugin fonctionnel T2 (plein), en distinguant les cas aboutis (bleu) des cas échoués (rouge).

La première observation majeure est que le prompt P2 est le seul niveau ayant systématiquement conduit à un plugin fonctionnel pour l'ensemble des plugins testés. À l'inverse, certains tests associés aux prompts P0 et P1 n'ont pas abouti, ces derniers étant entrés dans des boucles d'erreurs sans parvenir à produire un plugin conforme aux attentes.

On observe également que, pour deux des tests réalisés avec le prompt P2, le temps T2 est supérieur au temps T1. Cela indique que, bien que le premier build ait été atteint rapidement, au moins une phase de correction a été nécessaire avant d'obtenir un plugin pleinement fonctionnel. Ce comportement reste maîtrisé et ne remet pas en cause la robustesse globale de ce niveau de prompt.

L'analyse comparative des temps met par ailleurs en évidence que le prompt P1 apparaît comme le moins performant. On peut expliquer ces résultats par le fait que P1 guide partiellement l'IA dans une direction donnée, sans lui fournir un cadre suffisamment précis pour structurer correctement la logique du plugin. L'IA se retrouve alors contrainte par des indications incomplètes, ce qui limite sa capacité à converger vers un plugin stable.

À l'inverse, le prompt P0, bien que minimaliste, montre des performances plutôt correctes pour des plugins relativement simples. Dans ces cas, le faible niveau de contrainte semble laisser à l'IA une liberté suffisante pour explorer par elle-même une solution fonctionnelle. Cette observation suggère que deux approches peuvent être efficaces : soit fournir un prompt très structuré et détaillé, soit au contraire laisser une grande liberté à l'IA, mais qu'une approche intermédiaire mal définie est moins favorable.

Le graphique montre également qu'il n'existe pas de relation strictement linéaire entre la complexité du plugin et le temps de développement. Pour certains plugins jugés plus simples à coder, les temps restent comparables quel que soit le niveau de prompt. En revanche, pour des plugins plus complexes, les prompts structurés de type P2 permettent de réduire le temps de développement en évitant à l'IA d'explorer plusieurs logiques de programmation successives.

De manière générale, malgré le nombre limité de plugins testés, les résultats indiquent que le temps de développement assisté par l'IA se situe majoritairement entre 5 et 20 minutes. Il convient toutefois de rappeler qu'un plugin abouti au sens fonctionnel n'est pas nécessairement optimal en termes de qualité globale, ce qui justifie l'analyse complémentaire utilisant les 8 paramètres de notation du plugin définis au chapitre précédent.

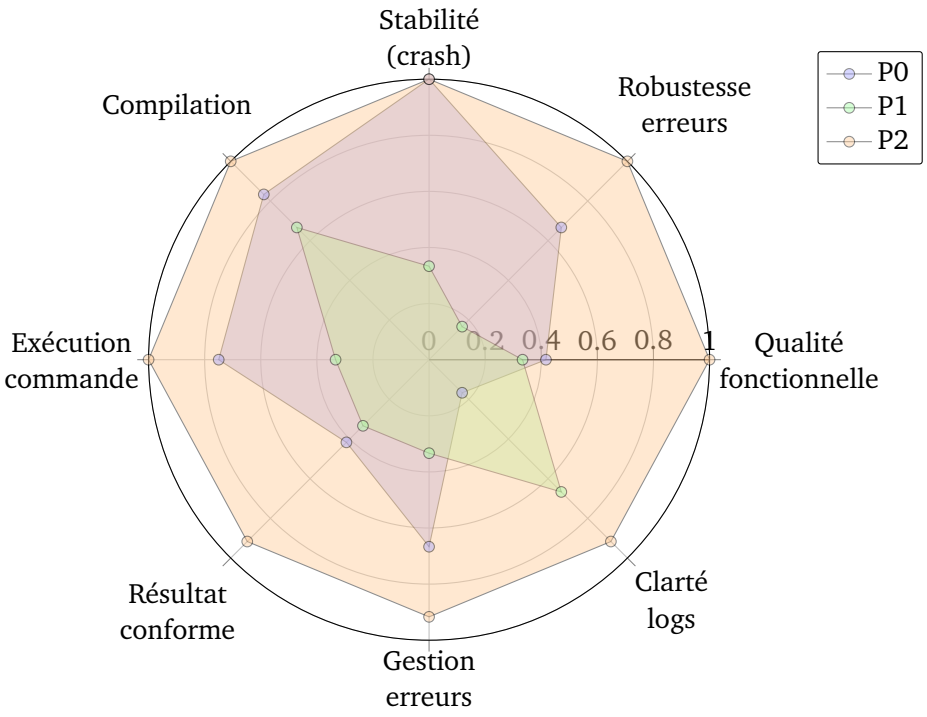


FIGURE 3.2 – Performance moyenne du processus selon les niveaux de prompt

Cette seconde figure représente les performances moyennes des plugins générés grâce aux huit critères d'évaluation, afin d'obtenir une note globale par niveau de prompt. Ce graphique radar permet non seulement d'évaluer la capacité de l'IA à produire un plugin fonctionnel, mais également la qualité du plugin.

Là encore, le prompt P2 se distingue nettement, avec un score quasiment maximal sur l'ensemble des critères évalués. Ce graphique confirme bien l'observation précédente selon laquelle un prompt structuré et détaillé permet à l'IA de générer des plugins de meilleure qualité.

À l'inverse, le prompt P1 apparaît comme le moins performant sur la majorité des critères, malgré une bonne qualité des messages affichés à l'utilisateur.

Le prompt P0, quant à lui, permet d'obtenir des plugins fonctionnels, mais montre des limites sur les aspects liés à l'interface utilisateur et à la clarté des retours : ces éléments n'ayant pas été explicitement guidés dans le prompt, il est logique que l'IA n'y ait pas prêté une attention particulière.

Enfin, l'analyse quantitative met en évidence que, lorsqu'une correction est nécessaire suite à une erreur d'exécution ou de compilation, trois retours de messages d'erreur sont souvent nécessaires avant d'obtenir un résultat satisfaisant. Il est difficile de tirer une conclusion définitive sur ce point. Toutefois, les corrections effectuées par l'IA restent généralement rapides, de l'ordre d'une à deux minutes par itération, ce qui limite l'impact global sur le temps total de développement.

Cette analyse quantitative met ainsi en évidence que P2 est le niveau de prompt optimisé pour le développement de plugins Civil 3D assisté par l'IA, suivi par P0. P1 apparaît comme le prompt le moins efficace. Cette analyse quantitative constitue une base objective pour l'analyse qualitative et la discussion des limites présentées juste après.

3.2.2 Analyse qualitative des résultats

Au-delà des indicateurs quantitatifs présentés précédemment, l'analyse qualitative des essais permet de mieux comprendre les mécanismes expliquant les succès et les échecs observés lors du développement assisté par l'IA. Cette analyse s'appuie principalement sur les commentaires notés lors de chaque test, tant du point de vue du développeur que de l'utilisateur (voir Annexe A.3).

Influence du wording et compréhension de la logique métier La première observation que l'on peut faire concerne l'importance du choix des mots utilisés dans la description fonctionnelle du plugin. Lors des essais sur le premier plugin, une mauvaise formulation a conduit l'IA à suivre une logique erronée : notre prompt indiquait la génération d'une polyligne à partir d'un fichier CSV, puis sa transformation en axe. Or, dans la logique de Civil 3D, ce ne sont pas les polylignes elles-mêmes qui deviennent des axes, mais les entités géométriques qui les composent.

Ce décalage entre la formulation que l'on utilise et la réalité du fonctionnement logiciel conduit Cursor à chercher des solutions inexistantes ou inadaptées, et empêche la production du plugin. Ce comportement illustre pleinement le concept de *wording*, largement décrit dans la littérature, selon lequel un mauvais choix de termes peut orienter l'IA vers une direction technique incorrecte sans qu'elle explore d'alternatives viables (CHEN et al., 2025).

On en conclut qu'en cas d'incertitude, il est préférable de rester volontairement plus vague plutôt que de transmettre une indication précise mais incorrecte qui fait entrer l'IA dans des boucles d'erreurs. La consultation préalable de la documentation Autodesk apparaît ici comme un prérequis essentiel.

Maîtrise différenciée des types de traitements La seconde observation concerne la nature des traitements réalisés par le plugin. Lors des essais, on remarque que Cursor se révèle particulièrement performant pour les tâches liées à l'analyse de texte et à la lecture de fichiers. La lecture et le traitement de fichiers du plugin 1 ou l'analyse de fichiers DWG et l'extraction d'informations plugin 3 ont été facilement et rapidement codés, avec peu d'erreurs.

À l'inverse, les opérations impliquant la création ou la modification d'objets Civil 3D se sont révélées plus délicates. La création d'axes, notamment, mobilise un grand nombre de paramètres et de dépendances internes à Civil 3D comme la création d'un site, ce qui augmente la difficulté pour l'IA à produire un code correct. Cette difficulté apparaît clairement dans les essais du plugin 1, où la gestion des sites, des axes et des entités associées a nécessité plusieurs retours de messages d'erreur.

Enfin, les fonctionnalités davantage héritées d'AutoCAD, telles que la création de polylignes ou l'insertion de blocs dans le plugin 2, se sont révélées simples à coder pour Cursor. Ces fonctions, plus

génériques et moins dépendantes de la logique spécifique à Civil 3D, semblent mieux maîtrisées par l'IA.

Fluidité d'utilisation et interaction avec l'utilisateur Les commentaires mettent également en évidence une différence dans la qualité de l'interaction avec l'utilisateur selon le niveau de prompt utilisé. Lorsque le prompt est structuré et détaillé, l'IA prend en compte les aspects liés à l'ergonomie et à la clarté des échanges. Les plugins générés dans ce cadre fournissent des messages explicites, des récapitulatifs utiles dans la ligne de commande et des indications claires sur l'état du traitement en cours.

À l'inverse, pour les prompts moins structurés (P0 et surtout P1), l'IA accorde peu d'attention à cet aspect. Les tests ont révélé des problèmes récurrents comme des messages incomplets, des erreurs de formulation, des problèmes gestion des caractères spéciaux (accents), l'absence de retours en cas d'erreur ou des instructions peu claires pour l'utilisateur. Cette différence confirme que la qualité de l'interface utilisateur n'est pas spontanée, mais est corrélée au niveau de précision du prompt.

Reproductibilité et irrégularité des résultats Un autre point important concerne la reproductibilité des résultats. Suite à des tests utilisant les mêmes prompts, il a été obsevé que l'IA ne fournit pas systématiquement le même code ni la même structure de plugin d'un test à l'autre.

Le prompt P2 permet de réduire cette variabilité en encadrant davantage le raisonnement du modèle, mais ne garantit pas pour autant une reproductibilité parfaite de résultats.

Ce comportement irrégulier constitue une limite importante dans le contexte de développement logiciel, puisqu'il n'est pas possible d'assurer à 100 % l'obtention d'un résultat identique d'une exécution à l'autre. L'IA fonctionne ici comme un générateur probabiliste de solutions plausibles, et non comme un système déterministe.

Gestion des erreurs et création de solutions erronées Enfin, la dernière observation concerne le comportement de l'IA lorsqu'elle se retrouve dans une impasse. Après plusieurs retours d'erreurs successifs, Cursor tend parfois à générer des fonctions inexistantes. On peut expliquer ce comportement principalement par deux mécanismes. Soit l'utilisation d'informations issues de versions antérieures de l'API, soit la confusion avec des fonctions provenant d'autres logiciels proches.

Dans ces situations, l'IA tente de produire une solution « plausible » pour sortir de l'impasse, même si celle-ci n'est pas techniquement valide. Ce phénomène confirme que l'IA ne dispose pas d'une compréhension réelle du cadre métier et qu'un regard humain reste indispensable pour identifier les erreurs conceptuelles, recadrer le raisonnement et interrompre les boucles d'erreurs si nécessaire.

Cette analyse qualitative met ainsi en évidence que l'intelligence artificielle constitue un puissant assistant au développement, à condition d'être utilisée avec discernement. La connaissance minimale des logiques métier, des limites de l'API et des possibilités offertes par le logiciel reste un facteur déterminant dans le développement assisté par l'IA.

3.3 Apports de l'étude

Cette dernière partie vise à synthétiser les apports principaux de cette étude. Elle s'appuie sur les résultats expérimentaux obtenus et sur l'analyse détaillée présentée dans les sections précédentes.

3.3.1 Apports méthodologiques

Le premier apport de cette étude réside dans la mise au point d'une méthode d'écriture de prompt structurée, complète et optimisée pour le développement de plugins Civil 3D assisté par l'IA. À l'issue du protocole expérimental et de l'analyse des résultats, la méthode P2 s'est imposée comme la plus pertinente et la plus fiable parmi les trois niveaux testés.

Cette méthode repose sur une structuration explicite du prompt, intégrant successivement le contexte, l'objectif fonctionnel, les entrées et sorties attendues, les étapes de traitement, les contraintes techniques et les critères de validation. Elle a pu être progressivement améliorée grâce aux retours fournis par l'IA elle-même, notamment via la génération d'un fichier de retour dédié (*"Prompt optimisé"*), permettant d'identifier les informations superflues, ambiguës ou manquantes.

Les résultats confirment que notre approche permet de canaliser efficacement le raisonnement de l'IA, de réduire les interprétations implicites que l'on peut retrouver dans P0 et P1, et de limiter les boucles d'erreurs. Elle constitue ainsi une méthode de rédaction générale réutilisable pour le développement assisté par l'IA, au-delà des cas étudiés dans ce projet.

3.3.2 Apports en termes de productivité

Un apport majeur de cette étude est la confirmation de la faisabilité du développement de plugins Civil 3D assisté par l'IA. Les essais réalisés montrent qu'il est possible de générer des plugins fonctionnels, robustes et exploitables, capables d'améliorer significativement les chaînes de travail sous Civil 3D.

Le gain de productivité est double. D'une part, les plugins développés permettent d'automatiser des tâches répétitives ou complexes, réduisant ainsi le temps de traitement dans les projets d'ingénierie civile. D'autre part, le processus même de création des plugins est fortement accéléré.

Une comparaison a été menée entre le développement assisté par l'IA (en utilisant le prompt P2) et un développement classique réalisé par un développeur expérimenté, en considérant des hypothèses optimistes sur la vitesse de frappe et la maîtrise métier d'un développeur. Cette comparaison est illustrée dans la figure suivante, qui détaille les différentes étapes du développement selon les deux méthodes sur les différents plugins.

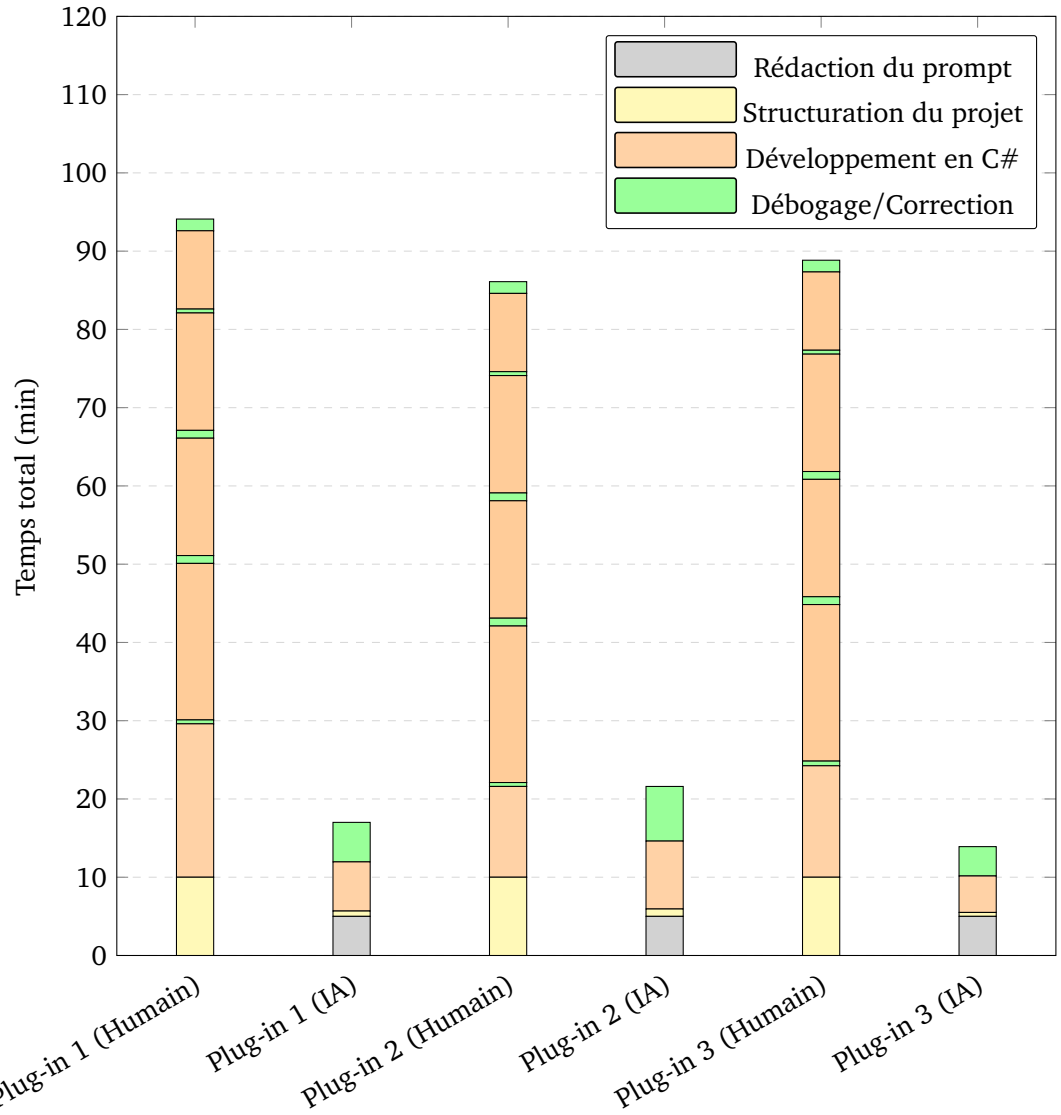


FIGURE 3.3 – Comparaison de temps de développement entre un humain et l'IA basée sur les prompts de niveau P2

Cette figure illustre une comparaison entre une estimation du temps nécessaire à un développeur humain expérimenté pour réaliser chaque plugin et le temps mesuré pour l'IA en utilisant le prompt P2.

L'estimation humaine s'est faite suite à la recherche, sur de multiples blogs et forums, de la vitesse moyenne de développement d'un développeur expérimenté. Nous nous sommes donc basés sur une vitesse d'écriture de 3,6 caractères par seconde et une correction ou un débogage toutes les 100 lignes de code.

On obtient ainsi des temps de développement humain d'environ 1h30 pour nos plugins. Suite à des discussions avec (BERSON, 2025), nous en avons conclu que ces estimations sont très optimistes, car elles supposent une maîtrise parfaite du sujet et une absence totale de temps d'attente ou de recherche. Les temps réels de développement humain seraient donc plus de l'ordre de une à deux journées complètes.

Même dans ce cadre favorable au développement manuel, un gain de temps significatif est observé, de l'ordre de 80% (soit une heure et douze minutes) en se basant sur les temps moyens mesurés.

Ces résultats confirment donc que l'IA constitue un levier puissant pour accélérer la mise en production de plugins Civil 3D, tout en conservant un niveau de qualité satisfaisant lorsque le prompt est correctement structuré.

3.3.3 Accessibilité du développement de plugins

Un autre apport important de ce travail concerne l'accessibilité du développement de plugins Civil 3D. Traditionnellement réservé à des profils fortement spécialisés en C# et en API Autodesk, ce type de développement devient plus abordable grâce à l'assistance de l'IA.

La méthodologie proposée permet à des utilisateurs avancés de Civil 3D, mais non experts en programmation, de concevoir et de faire évoluer leur propre logiciel selon leurs besoins. L'IA agit ici comme un intermédiaire entre le besoin métier et l'implémentation technique, à condition que l'utilisateur conserve une compréhension des logiques de développement et des limites de l'API.

3.3.4 Livrables et valorisation du travail

Enfin, un apport concret de cette étude est la production de notices complètes dédiées au développement de plugins Civil 3D assisté par l'IA. L'objectif de ces notices est de rendre cette méthode de développement directement exploitable par la communauté d'utilisateurs de Civil 3D.

Ces notices couvrent l'ensemble du processus à travers deux documents. Le premier se concentre sur l'intérêt des plugins Civil 3D, la mise en place de l'environnement de développement et les bonnes pratiques et erreurs à éviter (voir Annexe A.4). Le second document détaille l'utilisation des IA, l'impact du prompt dans le processus et la méthode d'écriture de prompt optimisée (P2), (voir Annexe A.5). Elles ont été conçues en collaboration avec Guillaume Berson, notamment dans la perspective d'une publication sur la plateforme Village BIM pour démocratiser l'utilisation de cette méthode.

Un soin particulier a été apporté à l'accessibilité et à l'esthétique du document. La charte graphique d'Autodesk a été reprise afin de proposer un support lisible, cohérent et professionnel, facilitant l'appropriation de ces nouvelles pratiques par les utilisateurs. La notice complète est fournie en annexe.

Cet ensemble de livrables permet ainsi de diffuser nos résultats auprès de la communauté concernée et de valoriser le travail réalisé dans le cadre de ce projet.

CHAPITRE

4

OUVERTURE ET DISCUSSION SCIENTIFIQUE

4.1 Bilan et perspectives d'amélioration

Les résultats obtenus confirment qu'un développement de plugins Civil 3D assisté par IA est opérationnel, à condition de maîtriser la communication avec le modèle. Dans notre cas, la structuration du prompt (méthode P2) joue un rôle de *contrat* entre le besoin métier et le raisonnement probabiliste du LLM, en limitant les interprétations implicites et en réduisant l'exploration de solutions non pertinentes. Cette logique rejoint plus largement l'idée que, lorsque l'on délègue une tâche à un système d'optimisation, la spécification de l'objectif et des contraintes devient un verrou critique : un objectif mal exprimé peut produire un résultat correct au sens strict, mais inadéquat au regard du besoin réel (BOSTROM, 2014).

Pour renforcer la portée scientifique de notre protocole, plusieurs pistes sont naturelles :

— Augmenter le nombre de plugins et diversifier les familles de fonctions d'API utilisées (surfaces, réseaux, profils en long, objets corridor) afin de tester la robustesse de P2 au-delà des trois cas d'étude.
— Introduire de nouveaux facteurs comme la taille du prompt, la densité de contraintes, le nombre d'étapes, le degré d'incertitude métier, et rechercher des corrélations avec les indicateurs de succès (T1/T2, itérations de build, score).
— Étudier la sensibilité des modèles en répétant chaque test plusieurs fois avec un prompt identique, afin d'estimer une dispersion (moyenne ± écart-type) et de distinguer *performance moyenne* et *stabilité*.

4.2 Apports potentiels de Spec Kit

Un point négatif ressort de notre étude : la non-reproductibilité stricte. À prompt identique, la solution générée peut varier, ce qui rend difficile la garantie d'un résultat fiable à coup sûr. Dans cette logique, des approches dites *spec-driven* deviennent pertinentes, car elles déplacent l'effort depuis la "bonne formulation" vers un artefact de spécification plus stable, vérifiable et partageable.

Le module *Spec Kit* proposé par GitHub s'inscrit précisément dans cette tendance : il vise à formaliser des exigences (spécifications) et à guider la production de livrables techniques à partir de ces

exigences, avec un accent mis sur la traçabilité et la réduction des ambiguïtés (GitHub, 2025). Dans un contexte comme le nôtre, l'intérêt est double :

- **Encadrement** : imposer une structure de sortie (fichiers attendus, règles de compatibilité, critères de validation), ce qui rapproche le prompt d'un cahier des charges exécutable.
- **Régularité** : réduire les différences entre les réponses en forçant l'IA à répondre à une spécification, plutôt qu'à improviser une architecture.

Une perspective directe serait donc de comparer l'utilisation de P2 pur vs P2 + Spec Kit, sur un jeu de plugins plus large, pour mesurer l'impact sur la stabilité, le taux d'échec et le nombre d'itérations.

4.3 Généralisation à d'autres logiciels Autodesk

Notre protocole est centré sur Civil 3D, mais une partie de la chaîne est transposable dès lors que l'environnement d'extension repose sur .NET et des DLL d'API. C'est notamment le cas de Revit (API .NET) et, plus largement, de plusieurs logiciels de la suite Autodesk. Une extension de l'étude consisterait à tester si la méthode P2 conserve ses bénéfices lorsque :

- L'API est plus fortement orientée objets architecturaux sur Revit.
- Les contraintes de transaction et de contexte document sont plus strictes (un projet de lotissement sur Revit).
- L'exécution exige davantage d'interactions avec l'utilisateur.

L'enjeu n'est pas uniquement la productivité : c'est aussi la construction d'un *standard* de rédaction de prompts, réutilisable d'un logiciel à l'autre.

4.4 Communication avec l'IA

Notre retour d'expérience (poids du wording, risques de fausses pistes, boucles d'erreurs) s'inscrit dans une problématique plus large décrite dans la littérature : les systèmes d'IA sont des optimiseurs, et la difficulté majeure devient la *traduction* d'un objectif humain vers une mission que doit effectuer l'IA.

J'ai découvert cet enjeu dans une vidéo documentaire d'(EGO, 2025) qui illustre ce point via l'expérience de pensée du "paperclip maximizer" (popularisée par (Bostrom, 2014)) : un objectif simple, mal borné, peut conduire à des stratégies extrêmes dès lors que le système gagne en capacité (Bostrom, 2014). Sans aller vers des scénarios dystopiques, on observe déjà des comportements de contournement : des modèles qui optimisent leurs résultats sans aucune limite éthique, ou des modèles dans *The AI Scientist* qui modifient leur environnement d'exécution pour contourner des contraintes expérimentales (Fu et al., 2024). Ces exemples convergent vers une idée simple : plus l'agent est capable, plus l'écart entre objectif écrit et objectif voulu devient coûteux.

Dans les systèmes conversationnels simples comme ChatGPT ou Gemini, cette difficulté se manifeste sous une forme plus facile à gérer : hallucinations, fonctions inventées, confusion d'API, ou justification persuasive d'une solution incorrecte. La revue de (Zhou et al., 2024) synthétise ces risques de comportements de tromperie qu'utilisent les modèles, et discute des pistes d'atténuation de ces comportements que l'on peut adopter. La question n'est donc pas uniquement "est-ce que l'IA peut aider ?", mais "comment garder le contrôle ?".

Deux verrous restent majeurs :

- **Limites du feedback humain.** Les méthodes de type RLHF (technique de feedback humain pour améliorer les performances du modèle) améliorent le comportement observable, mais présentent des limites structurelles qui motivent encore une recherche active afin de sécuriser l'utilisation de cette nouvelle technologie (Lambert et al., 2023).
- **Opacité des modèles.** Même lorsque les sorties semblent correctes, comprendre "pourquoi" le modèle a choisi telle solution est difficile. Ce côté "boîte noire" limite la confiance que l'on

peut avoir envers les modèles. Les travaux sur l'interprétabilité progressent, mais l'écart entre la vitesse de développement des modèles et le budget mis dans la recherche de compréhension des IA demeure un point de tension (BUBECK et al., 2023).

4.5 Impact sur notre métier

Pour les métiers liés à la topographie et au génie civil, l'enjeu n'est pas la disparition du raisonnement technique, mais son déplacement vers des compétences orientées vers les outils d'IA. L'IA va accélérer tous nos processus de traitement (dessin de plan, traitement de nuages de points, modélisation 3D), il faudra donc, de notre côté, développer des compétences dans la compréhension de ces outils afin de garder un contrôle sur cette méthode de travail. On peut penser à une évolution du métier vers plus de supervision de productions automatiques. Bien évidemment, ce changement n'est pas pour demain, mais le développement de ces outils est exponentiel, et il est important de se préparer à cette potentielle évolution.

Aussi, on peut penser à l'impact que cette chaîne de traitement assistée par IA peut avoir sur le marché du travail. En effet, si l'IA permet de produire plus rapidement des livrables techniques, cela peut impacter la demande dans notre secteur. Il y aurait besoin de moins d'ingénieurs et plus de techniciens pour relever les données terrain. Évidemment, de nouveaux corps de métiers vont aussi émerger autour de l'IA, comme des spécialistes en prompt engineering, ou des superviseurs de chaînes de traitement IA. Je suis content de pouvoir être acteur de ce changement plutôt que spectateur et potentiellement être dépassé par cette révolution technologique.

CONCLUSION

L'objectif principal de ce projet était d'évaluer l'apport réel de l'intelligence artificielle dans le développement de plugins Civil 3D, en particulier à travers l'utilisation d'un IDE assisté par IA, et de comprendre dans quelles conditions cette assistance devient pertinente. Au-delà de la simple question *« l'IA peut-elle coder un plugin Civil 3D ? »*, ce travail visait surtout à identifier une méthode fiable, reproductible et exploitable.

Les expérimentations menées montrent que la génération de plugins fonctionnels par IA est aujourd'hui possible, mais qu'elle dépend fortement de la manière dont le besoin est formulé. Le protocole mis en place, reposant sur trois niveaux de prompt, a permis de mettre en évidence une tendance claire : plus le prompt est structuré et précis, plus le résultat est robuste, cohérent et proche des attentes métier. Le prompt de type P2, construit autour d'un contexte clair, d'objectifs explicites, d'entrées et sorties définies, d'étapes de traitement ordonnées et de critères de validation, s'est révélé être le plus performant sur l'ensemble des plugins testés.

L'analyse quantitative a montré que cette structuration permet non seulement d'augmenter le taux de réussite des développements, mais également de réduire le temps global nécessaire pour obtenir un plugin fonctionnel. L'analyse qualitative a, quant à elle, mis en évidence des aspects plus subtils, tels que l'importance du choix des mots, la gestion des erreurs, la clarté des messages utilisateur ou encore la stabilité du comportement généré par l'IA.

Ce projet confirme ainsi que l'intelligence artificielle permet aujourd'hui de développer des plugins Civil 3D de manière efficace, à condition de bien l'encadrer par des prompts adaptés. Elle permet de réduire la barrière technique du développement de plugins, d'augmenter la productivité et d'ouvrir l'automatisation à des profils d'utilisateurs qui n'auraient pas, initialement, les compétences en programmation nécessaires. Toutefois, la connaissance métier, la compréhension des API et la capacité à formuler un besoin précis restent indispensables pour garantir des résultats fiables.

POSTFACE

Sur un plan plus personnel, ce projet a profondément modifié ma perception de l'intelligence artificielle. Ce travail a confirmé mon intérêt pour les enjeux actuels liés à l'intelligence artificielle, tant sur le plan technique que sur le plan plus global de la relation homme–machine. La capacité à dialoguer efficacement avec une IA, à encadrer son comportement et à comprendre ses limites me semble être l'un des défis majeurs des années à venir. Dans ce contexte, l'ingénieur de demain ne sera pas seulement un utilisateur d'outils intelligents, mais un acteur capable de guider, contrôler et exploiter ces technologies de manière responsable.

Je suis également fier d'avoir pu réaliser un projet de recherche avec des attentes professionnelles concrètes. C'est un premier pas vers le monde de la recherche et un pas de plus vers le monde du travail. J'aborde la prochaine étape de mon parcours, qui est le PFE, avec une vision plus claire des enjeux et méthodes de travail derrière un projet de recherche.

TABLE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ALGORITHMES

BIBLIOGRAPHIE

Références bibliographiques et revues scientifiques

BECKER, Joel et al. (2025). “Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity”. In : Mesure quantitative de l’impact des outils d’IA sur la productivité des développeurs expérimentés.

BOSTROM, Nick (2014). *Superintelligence : Paths, Dangers, Strategies*. Oxford University Press.

BUBECK, Sébastien et al. (2023). “Sparks of Artificial General Intelligence : Early experiments with GPT-4”. In : *arXiv preprint arXiv :2303.12712*. https://arxiv.org/abs/2303.12712.

CHEN, Banghao et al. (2025). “Unleashing the potential of prompt engineering for large language models”. In : Analyse du comportement des grands modèles de langage en fonction du prompt.

FU, Y. et al. (2024). “The AI Scientist : Towards Fully Automated Open-Ended Scientific Discovery”. In : *arXiv preprint arXiv :2408.06292*. https://arxiv.org/abs/2408.06292.

HAJIPOUR, Hossein (2024). “AI Code Generation Models : Opportunities and Risks”. In : Analyse des opportunités et risques liés aux modèles d’IA pour la génération de code.

LAMBERT, Nathan et al. (2023). “Open Problems and Fundamental Limitations of Reinforcement Learning from Human Feedback”. In : *arXiv preprint arXiv :2307.15217*. https://arxiv.org/abs/2307.15217.

PARTHASARATHY, Venkatesh Balavadhani et al. (2024). “The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs”. In : Revue exhaustive des technologies et bonnes pratiques pour le fine-tuning de LLM.

RAMPF, Felix (2017). “Development of an IFC Alignment Import/ Export Plug-In for Autodesk AutoCAD Civil 3D”. In : Principes et bonnes pratiques pour structurer et tester un plugin Civil 3D.

RAY, Partha Pratim (2025). “A Review on Vibe Coding : Fundamentals, State-of-the-art,Challenges and Future Directions”. In : Revue des logiciels et techniques de développement piloté par langage naturel assisté par IA.

SERGEYUKA, Agnia et al. (2024). “Using AI-Based Coding Assistants in Practice : State of Affairs, Perceptions, and Ways Forward”. In : Utilisation actuelle et perception des assistants de code basés sur l’IA par les développeurs.

THAW, Thin Thu Thu (2025). “How Effective are AI-powered Code Assistants in Enhancing Developer Productivity ?” In : Analyse qualitative du gain de productivité lié à l’utilisation d’assistants de code basés sur l’IA.

ZHOU, Y. et al. (2024). “AI deception : A survey of examples, risks, and potential solutions”. In : *Patterns*. https://pmc.ncbi.nlm.nih.gov/articles/PMC11117051/.

Documentation logiciels

AUTODESK (2025a). *Autodesk Civil 3D 2025 .NET API Developer Guide*. https://help.autodesk.com/view/CIV3D/2025/ENU/?guid=GUID-DA303320-B66D-4F4F-A4F4-9FBBEC0754E0.

AUTODESK (2025b). *Autodesk Civil 3D 2025 .NET API Reference Guide*. https://help.autodesk.com/view/CIV3D/2025/ENU/?guid=GUID-DA303320-B66D-4F4F-A4F4-9FBBEC0754E0.

CURSOR (2025). *Documentation sur le fonctionnement de Cursor*. https://cursor.com/docs.

GITHUB (2025). *Spec Kit (spec-driven development toolkit)*. https://github.com/github/spec-kit.

MICROSOFT (2025). *Documentation sur le SDK .NET*. https://learn.microsoft.com/fr-fr/dotnet/fundamentals/.

Autres

BERSON, Guillaume (2025). *Discussions professionnelles*. Notes d’échanges et retours d’expérience.

EGO (2025). *L’horreur existentielle de l’usine à trombones.* https://www.youtube.com/watch?v=ZP7T6WAK3Ow.

ANNEXE

A

ANNEXES

A.1 Prompt environnement

Les éléments entre crochets [ ] sont spécifiques à notre environnement/projet.

Contexte environnement de travail

- OS : [Windows 11]
- Terminal par défaut : PowerShell
- .NET SDK installé : [.NET 8.0]
- AutoCAD : [2026] ; Civil 3D : [2026]

Contexte de développement

- Racine du workspace : [`T:\Jean PELISSIER\PRT\Projet`]
- Tous les projets C# sont contenus dans cette racine.
- Chaque plugin : projet SDK-style (`Microsoft.NET.Sdk`) ciblant [`net8.0-windows`], plateforme [x64].
- Références AutoCAD/Civil 3D à ajouter dans chaque `.csproj` avec `<Private>False</Private>` :
  - AutoCAD DLLs : [`C:\Program Files\Autodesk\AutoCAD 2026\`]
    acdbmgd.dll, acmgd.dll, accoremgd.dll
  - Civil 3D DLLs : [`C:\Program Files\Autodesk\AutoCAD 2026\C3D\`]
    AeccDbMgd.dll, AeccLandMgd.dll, AeccRoadwayMgd.dll, AecBaseMgd.dll

Structure attendue

1. Création du projet :
   - Dossier : [`T:\Jean PELISSIER\PRT\Projet\Plugin`]
   - Fichiers attendus : [`Plugin.csproj`] (références AutoCAD/Civil 3D), dossier `/Commands` avec le code, `/Properties/AssemblyInfo.cs`, `.editorconfig`, `.gitignore`, `.vscode/tasks.json`, `README.md`
2. Code principal :
   - Classe [`Commands`] dans [`/Commands/Plugin.cs`]
   - Gestion d'erreurs : `try/catch` + messages utilisateur clairs
   - Interactions utilisateur via `Autodesk.AutoCAD.ApplicationServices.Editor`
3. Compilation :
   - `dotnet restore`
   - `dotnet build -c Release -p:Platform=x64`
   - Sortie attendue : [`bin\x64\Release\net8.0-windows\Plugin.dll`]
   - Aucun lancement automatique de Civil 3D

4. Objectif final :
   — Build totalement automatique
   — Fichiers structurés pour tests/évolutions
   — Cursor corrige les erreurs jusqu'à "Build succeeded"
   — Pas de lancement externe de Civil 3D ; usage exclusif de `dotnet` CLI
   — Chaque projet est autonome avec ses références dans le `.csproj`
5. Fichier "Prompt Optimisé.txt"
   — Emplacement : dossier du plugin ([`T:\Jean PELISSIER\PRT\Projet\Plugin`])
   — Nom exact : [`Prompt Optimisé.txt`] (UTF-8, texte brut)
   — Contenu : version optimisée du prompt
   — Annexe : 3 chemins d'amélioration du prompt

A.2 Prompt P2 - Exemple sur le plugin de polyligne

Contexte

— Tu es un développeur C# expert en API AutoCAD/Civil 3D.
— Tu développes un plugin Civil 3D 2026.
— Le plugin doit être autonome, sans dépendances externes.
— Le code doit être clair, structuré et commenté.

Objectif

Créer un plugin Civil 3D permettant à l'utilisateur de créer une polyligne 2D à partir de deux points, définis soit par sélection dans le dessin, soit par saisie manuelle de coordonnées.

Fonctionnalité attendue

— proposer à l'utilisateur un mode de définition des points : sélection graphique dans le dessin ou saisie des coordonnées (X, Y) au clavier ;
— créer une polyligne 2D reliant les deux points définis ;
— afficher un message de confirmation dans la ligne de commande.

Entrées

— Mode de saisie : sélection de points dans le dessin, ou saisie manuelle des coordonnées X et Y.
— Point 1 : premier point défini par l'utilisateur.
— Point 2 : second point défini par l'utilisateur.

Sorties

— Une polyligne 2D ajoutée au ModelSpace de Civil 3D, reliant les deux points.
— Un message indiquant que la polyligne a été créée avec succès.

Étapes de traitement

1. Demander à l'utilisateur de choisir le mode de saisie (sélection ou coordonnées).
2. Selon le mode choisi :
   — soit demander la sélection graphique des deux points ;
   — soit demander la saisie numérique des coordonnées X et Y pour chaque point.
3. Vérifier la validité des points saisis.
4. Créer une entité `Polyline` :
   — ajouter les deux points comme sommets ;
   — définir la polyligne comme ouverte.
5. Ajouter la polyligne au ModelSpace à l'aide d'une transaction.
6. Valider la transaction.
7. Afficher un message de confirmation dans la ligne de commande.

Contraintes techniques

— Utiliser uniquement les classes de l'API AutoCAD (`Polyline`, `Point3d`, `Transaction`, etc.).
— Encadrer toutes les modifications du dessin dans une transaction.
— Ne pas créer de dépendances externes.
— Garantir la compatibilité avec Civil 3D 2026.

Gestion des erreurs

— Si l'utilisateur annule une saisie ou fournit des coordonnées invalides, interrompre proprement la commande.
— Ne pas créer la polyligne si les deux points ne sont pas valides.
— Afficher des messages d'erreur clairs dans la ligne de commande.

Critères de validation

— Le projet compile sans erreur et génère une DLL fonctionnelle.
— La commande est reconnue dans Civil 3D après chargement.
— La polyligne est correctement créée quel que soit le mode de saisie choisi.
— Aucun comportement inattendu ou crash ne survient à l'exécution.

A.3 Résultats bruts

Plug-in	Prompt	T1 (premier build)	T2 (version finale)	Err. build	Err. exéc.	Qualité	Robuste	Stabilité	Compil.	Exécution	Résultat	Gest. err.	Clarté logs	Score /8	Commentaire développeur	Commentaire utilisateur	Taille (ko)
Plug-in 1	P0	5'20" ; 6'25"	6'25"	4 ;2	1	0	1	1	0,5	0	1	0,5	0,5	4	Premier build compliqué au vu des 4 erreurs, puis erreur de la fonction, car il faisait un appel à un site Civil 3D où il y avait des informations sur l'alignement. 2e build OK : il n'arrivait toujours pas à créer un alignement correct mais il a créé la polyligne avec les instructions sur comment faire l'alignement.	Le plugin ne correspond pas à ce que je veux mais les instructions sur comment faire l'alignement sont claires.	313
Plug-in 1	P1	5'03" ; 6'10" ; 9'06" ; 10'01"	/	1 ;0 ;3 ;2	3	0	0	0	1	0	0	0	1	2	Après 4 essais, Cursor fait les mêmes erreurs, même en lui donnant des éléments de réponse. Erreur : not the same database. J'ai l'impression que Cursor a compilé 4 fois en changeant les mêmes choses en boucle. Je n'arrive pas à comprendre l'erreur tout seul.	J'ai l'impression que Cursor a compilé 4 fois en changeant les mêmes éléments en boucle. Je n'arrive pas à comprendre l'erreur tout seul.	314
Plug-in 1	P2	5'25" ; 6'10" ; 9'23" ; 11'20"	11'20"	3 ;0 ;0 ;2	3	1	1	1	1	1	1	1	1	8	Dans ce prompt, il y a une précision sur la création d'un site. Il a donc pris la création d'un site en compte s'il n'est pas créé. Même s'il a fallu 3 erreurs pour y arriver, c'est bon. D'après ce que j'ai compris, avec la formulation, il a compris qu'il fallait sélectionner une polyligne pour la faire devenir un alignement et il cherchait cette fonction qu'il ne trouvait pas. Lorsque je lui ai expliqué concrètement ce que j'avais et que je voulais que ma polyligne soit un élément de mon axe, ça lui a donné de bonnes indications pour qu'il utilise la bonne fonction. Conclusion : le choix des mots dans notre prompt peut totalement le bloquer et l'amener à chercher dans une direction qui n'est pas bonne. Lui laisser plus de liberté (Prompts 0 et 1) ne l'a pas aidé non plus. Donc une connaissance minimale de Civil 3D est nécessaire pour utiliser ça.	L'interface est super bien faite : on a une précision sur tout : sélection du fichier CSV, lecture du fichier, points valides trouvés : 51, Zmin = 150.000, Zmax = 160.000, tri des points par PK, gestion des doublons, création de la polyligne et de l'alignement, aucun site trouvé, création du site SITE_CSV, site créé avec succès : (1995639478672), polyligne créée avec 51 points, alignement 'ALN_CSV' créé. Polyligne disponible dans le dessin. Alignement 'ALN_CSV' créé avec succès, === Terminé avec succès ===, Nombre de points valides : 51, Zmin = 150.000, Zmax = 160.000, Site : SITE_CSV, Alignement : ALN_CSV, Temps d'exécution : 2900 ms.	322
Plug-in 2	P0	12'56" ; 15'03"	15'03"	3	1	1	1	1	1	1	1	0	1	7	Lors de la première version, il m'obligeait à sélectionner un bloc affiché dans le dessin, ce qui n'était pas le but du plugin, et il n'arrivait pas à afficher correctement les blocs sur l'alignement suite à la sélection d'un bloc. Lors de la deuxième version, après avoir retourné le message d'erreur plus ma demande de juste écrire le nom d'un bloc pour le choisir, ça marche parfaitement. La fonction marche bien, mais les retours dans la commande ne sont pas extrêmement clairs. Il y a des erreurs sur les accents et la formulation n'est pas très claire. Totalement utilisable pour autant.	La fonction marche bien, mais les retours dans la commande ne sont pas extrêmement clairs. Il y a des erreurs sur les accents et la formulation n'est pas très claire. Totalement utilisable pour autant.	303
Plug-in 2	P1	15'24"	15'24"	4	0	1	0,5	1	1	1	1	1	1	7,5	Résultat parfait dès la première exécution. Mais il y a un temps de réflexion assez long. Ajout comparé à la fonction précédente du choix de l'angle de notre bloc et décalage le long de l'alignement. Ça rajoute un énorme plus. Le petit problème est qu'il a mis une option "Pas d'orientation particulière" pour le bloc et celle-ci n'a pas de sens et elle ne marche pas.	Plugin avec des instructions claires. Point négatif : certaines options ne marchent pas. La présentation des résultats dans la ligne de commande est réussie.	310
Plug-in 2	P2	15'41"	15'41"	2	0	1	1	1	1	1	1	1	1	8	Résultat parfait dès la première exécution. Temps de réflexion un peu long. Réactions aux erreurs utilisateur parfaites. Il fait des vérifications sur toutes nos entrées et signale lorsqu'il y a un problème. Seul bémol : si on a sélectionné un nom de bloc qui n'existe pas, il le dit à la fin ; il pourrait prévenir avant. Le plugin est fluide, sans temps de latence, et répond exactement au besoin. Compilé du premier coup, il marche aussi parfaitement.	Le plugin est fluide, sans temps de latence, et répond exactement au besoin. Compilé du premier coup, il marche aussi parfaitement.	324
Plug-in 3	P0	14'08"	14'08"	4	0	0,25	0	1	1	0,5	0,25	0	0	3	Dans un premier lieu, la fonction marche, car on a effectivement un CSV en sortie avec dedans un recensement des entités, etc. Pour autant, ce CSV n'est pas compréhensible du tout : on remarque qu'il a bien compté des entités dans chaque DWG, mais sans aucune précision dessus. En plus, on doit renseigner à la main le chemin d'accès du dossier qu'on veut qu'il étudie, ce qui n'est pas pratique du tout.	Comme le chemin est écrit à la main, ce n'est pas fluide, et le CSV ne contient pas de précisions particulières. Il n'a pas pris d'initiatives sur le contenu du CSV ni sur l'analyse du fichier.	284
Plug-in 3	P1	6'08" ; 8'02" ; 9'56"	/	1 ;2 ;1	3	0	0	0	0	0	0	0	0	0	Même en lui expliquant plusieurs fois que la commande, même chargée, n'était pas disponible, il dit ne pas avoir accès à des DLL de Civil 3D que je n'ai pas : AeccLandMgd.dll. Après plusieurs explications, il n'y arrive toujours pas. Pourtant, le fichier C# paraît correct à la lecture.	Il n'est pas possible de tester la fonction.	265
Plug-in 3	P2	6'15" ; 8'25"	8'25"	0 ;0	0	1	1	1	1	1	0,75	0,75	0,75	7,25	Lors de la première exécution, le rendu CSV était bien détaillé et comprenait toutes les informations qu'on avait demandées. Le problème, c'est que c'est illisible au vu de la quantité de calques par défaut dans les gabarits Civil 3D, etc. Aussi, je devais écrire à la main les chemins d'accès pour les dossiers. Donc j'ai fait une demande pour améliorer le CSV en un fichier texte bien lisible, où on comprend bien tout ce qu'il y a, et j'ai demandé des fenêtres pour sélectionner les dossiers et l'emplacement du fichier texte. Il est facile à utiliser et très rapide. Le fichier texte produit correspond bien à ce qu'on attend. On pourrait avoir plus de détails si besoin, par exemple sur les sites, alignements, etc.	Il est facile à utiliser et très rapide. Le fichier texte produit correspond bien à ce qu'on attend. On pourrait avoir plus de détails si besoin, par exemple sur les sites, alignements, etc.	352

TABLE A.1 – Synthèse des essais par plug-in, niveau de prompt et critères d'évaluation.

A.4 Notice d'installation, utilisation et bonnes pratiques

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES STRASBOURG

AUTODESK
Make Anything

Mettre en place un plugin Civil 3D / AutoCAD avec l'IA

Notice d'installation, utilisation et bonnes pratiques

Rédacteur : PELISSIER Jean
Relecture : BERSON Guillaume

Janvier 2026

Pourquoi développer un plugin ?

Automatiser des tâches répétitives

Réduire le temps passé sur des actions manuelles répétitives et limiter les erreurs humaines en automatisant vos opérations longues et sensibles.

Outils adaptés à vos besoins

Créer des fonctionnalités parfaitement adaptées à vos besoins métiers et propres à vos méthodes de travail, afin d'intégrer vos calculs, contrôles et méthodes internes directement dans Civil 3D.

Méthodes unifiées

Garantir une méthode de travail identique pour tous, avec des paramètres choisis pour votre entreprise et des résultats reproductibles sur l'ensemble de vos projets.

Créer de la valeur interne

Développer des plugins permet de créer des outils durables qui renforcent l'expertise de l'équipe, soutiennent l'innovation et modernisent vos méthodes de production.

Comment crée-t-on un plugin Civil 3D ?

Autodesk met à disposition l'API complète de Civil 3D, accessible en C# via .NET.

Un développeur crée un projet .NET, programme les commandes nécessaires, compile une DLL et la charge dans Civil 3D via NETLOAD.

Ce processus demande du temps, une bonne connaissance de l'API et de la logique interne du logiciel.

Qu'est-ce que Cursor ?

Cursor est un environnement de développement (IDE) basé sur l'intelligence artificielle, pensé pour écrire du code avec vous.

Contrairement à un simple chatbot comme ChatGPT ou Gemini, Cursor a accès à vos fichiers, peut analyser vos projets, comprendre le contexte et créer ou modifier du code directement dans un éditeur.

L'expérience est fluide et intuitive : c'est du "vibe coding".

Suite à une discussion avec l'IA, elle peut :

- Créer un environnement pour votre plugin
- Coder en C# en utilisant l'API de Civil3D
- Compiler le code généré et build le DLL du plugin dans son terminal
- Corriger des erreurs dans un code jusqu'au bon fonctionnement de celui-ci.

Alternatives à Cursor

VSCode + Github Copilot

Rider + JetBrain AI

VSCode + Codex

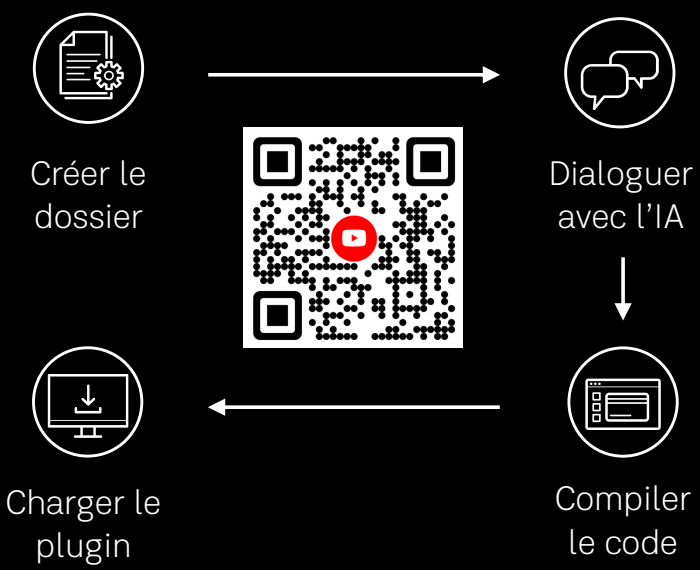
Pré-requis pour développer avec Cursor

Pour développer un plugin Civil 3D avec Cursor, il faut :

Une version Windows Civil3D/Autocad

.NET SDK installé pour compiler

Workflow simple



Bonnes pratiques

L'utilisation de Cursor n'est pas « magique », il existe des méthodes et des bonnes pratiques à suivre :

- Utiliser un prompt optimisé : La qualité de votre prompt détermine la qualité du plugin. Il est primordial de suivre les étapes décrites dans le document de rédaction de prompt. Cela passe par une méthodologie d'écriture précise. Vous pouvez d'ailleurs utiliser une autre IA pour rédiger ce prompt.
- Tester et faire des retours : Il est possible que le plugin ne soit pas fonctionnel du premier coup. Il est important de le tester dans Civil 3D et de retourner le message d'erreur, ou de préciser ce que vous souhaitez ajouter ou modifier.
- Avoir une structure de projet claire : Cursor est plus performant dans un environnement de projet organisé, avec des dossiers et sous-dossiers bien structurés.
- Configurer Cursor en mode semi-automatique : Il est possible d'activer une option semi-auto dans Cursor qui l'oblige à vous faire valider chaque étape manuellement, afin que vous gardiez un contrôle complet sur le développement du plugin.

Erreurs à éviter

Certaines erreurs qui peuvent paraître anodines ont tendance à faire entrer l'IA dans des boucles d'erreurs :

- Un "wording" imprécis ou mal choisi : Cursor suit la logique que vous décrivez. Un terme inadapté peut l'envoyer vers une mauvaise solution (ex : polyligne au lieu d'axe) et provoquer des erreurs en boucle. La précision du vocabulaire est essentielle.
- Penser que toutes les fonctions visibles dans Civil 3D existent dans l'API : Certaines commandes ou options disponibles dans l'interface peuvent ne pas exister dans l'API fournie par Autodesk. Cursor peut donc chercher des méthodes qui n'existent pas. Il faut garder à l'esprit que UI ≠ API.
- Oublier de donner à Cursor les éléments techniques indispensables :
    - Chemins vers les DLL Autodesk
    - La version de Civil 3D utilisée
    - La version du .NET SDK
    - La structure du projet

  Un plugin ne se décrit pas seulement par son objectif, il faut décrire l'environnement technique.

Conclusion

Créer des plugins Civil 3D peut donc être une bonne manière d'améliorer vos workflows, automatiser des tâches répétitives et gagner en efficacité sur vos projets. Cursor peut générer ces plugins de manière rapide et intuitive, à condition de s'appuyer sur une bonne connaissance métier et un prompt optimisé dont vous retrouverez l'explication dans la section suivante.

A.5 Notice de rédaction de prompts optimisés

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES STRASBOURG

AUTODESK
Make Anything

Développer des plugins Civil 3D assisté par l'IA

Notice de rédaction de prompts optimisés pour Cursor

BERSON Guillaume
PELISSIER Jean

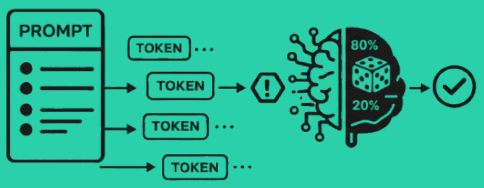
Janvier 2026

Qu’est-ce qu’un prompt ?

Le prompt n’est pas une simple question, mais une consigne qui guide tout le raisonnement que va suivre une IA.
Sa qualité conditionne directement la fiabilité et la pertinence du plugin généré.

Comment l’IA « comprend » un prompt

L’IA ne comprend pas le besoin humain.
Elle fonctionne en découpant le prompt en tokens (morceaux de mots) et en évaluant, à chaque étape, la probabilité du mot ou de l’action suivante.



Cursor analyse :

- Les mots utilisés
- leur ordre
- leur précision
- le contexte

Si une information est absente ou ambiguë, l’IA comble le vide en choisissant la solution qui est « statistiquement » la plus probable, ce qui peut l’amener à prendre une mauvaise direction.

Quel impact a-t-il sur le développement du plugin?

Temps de développement

Les prompts structurés dirigent l’IA dans la bonne direction dès le départ.
Cela évite les allers-retours inutiles liés aux incompréhensions, qui peuvent rallonger inutilement le temps de développement du plugin.

Qualité du code

Un prompt bien formulé favorise un développement clair et structuré.
Il conduit à un code mieux organisé, avec des commentaires pertinents et une séparation logique des fonctionnalités, facilitant ainsi la relecture et la compréhension par l’utilisateur.

Boucles d’erreurs

Si la logique du prompt n'est pas bien structurée, on s'expose à de possibles boucles d'erreurs dans lesquelles l'IA ne trouve pas ses erreurs et reste bloquée sur de fausses pistes. Cela donne donc un plugin même pas compilable.

Etapes de rédaction d'un prompt optimisé pour le développement de plugins Civil3D

1 Initialisation de l'environnement de développement

Avant de décrire la logique du plugin, il est nécessaire de demander à l'IA de structurer correctement l'environnement de développement.
Le prompt d'initialisation définit :

- L'environnement de travail : système d'exploitation, versions de Civil 3D et du SDK .NET.
- Les chemins de développement : racine du projet et accès aux bibliothèques Civil 3D (API).
- La structure attendue du projet :
    1. création du projet et des fichiers associés.
    2. organisation du code principal et gestion des erreurs.
    3. règles de compilation et résultat attendu (DLL).
    4. objectifs finaux et contraintes imposées à l'IA.

Sans un environnement défini, la génération et la compilation du plugin ne peuvent pas aboutir.

2 Contexte

On demande à l'IA de se comporter comme un développeur C# expert en plugins Civil 3D, chargé de concevoir un outil optimisant une mission précise.
Ce cadrage oriente ses choix techniques et sa logique de développement.

3 Objectif

Définir clairement le comportement attendu et l'objectif du plugin, en se concentrant sur le besoin métier plutôt que sur les aspects techniques.

4 Entrées

Expliciter les entrées fournies par le système ou l'utilisateur (fichiers, sélections, paramètres) et la manière dont le plugin les récupère.

7 Contraintes

Préciser les contraintes techniques et fonctionnelles à respecter (compatibilité Civil 3D, performances, gestion des erreurs), afin d'éviter que l'IA propose des solutions inadaptées.

6 Traitement

Décrire les principales étapes de traitement du plugin, dans un ordre logique, afin de guider le raisonnement de l'IA.
Éviter de décrire une logique de traitement si on ne la maîtrise pas

5 Sorties

Définir les sorties attendues du plugin (objets créés, messages, résultats visibles) et la forme sous laquelle elles sont produites dans Civil 3D.

8 Critères de validation

Définir des critères de validation clairs permettant de vérifier que le plugin répond correctement au besoin.
Ces critères portent sur le bon déroulement du traitement, la conformité du résultat obtenu, la clarté des messages affichés dans la ligne de commande.

Exemple : Prompt optimisé pour un plugin simple (création de polyligne)

CONTEXTE ENVIRONNEMENT DE TRAVAIL :

- OS : Windows 10 / 11
- Terminal par défaut : PowerShell
- .NET SDK installé : .NET 8 (dotnet CLI)
- AutoCAD version : 2026
- Civil 3D version : 2026

CONTEXTE DE DÉVELOPPEMENT :

- Racine principale du workspace : T:\Jean PELISSIER\PRT\Projet
- Tous les projets C# sont contenus dans cette racine.
- Chaque plugin doit être un projet SDK-style (Microsoft.NET.Sdk) ciblant net8.0-windows, plateforme x64.
- Les références AutoCAD/Civil 3D doivent être ajoutées directement dans chaque .csproj :
  - AutoCAD DLLs : C:\Program Files\Autodesk\AutoCAD 2026\ : acdbmgd.dll; acmgd.dll; accoremgd.dll
  - Civil 3D DLLs : C:\Program Files\Autodesk\AutoCAD 2026\C3D : AeccDbMgd.dll;AeccLandMgd.dll;AeccRoadwayMgd.dll ; AecBaseMgd.dll

Toutes les références doivent avoir <Private>False</Private> pour éviter la copie des DLLs.

STRUCTURE ATTENDUE :

1/ Création du projet :

- Nouveau dossier : T:\Jean PELISSIER\PRT\Projet\Plugin
- Créer un projet SDK-style net8.0-windows (C#, x64)
- Ajouter les fichiers nécessaires :
  - o Plugin.csproj avec toutes les références AutoCAD/Civil 3D configurées
  - o dossier /Commands avec le fichier principal de commande
  - o dossier /Properties avec AssemblyInfo.cs (optionnel mais recommandé)
  - o .editorconfig, .gitignore
  - o .vscode\tasks.json pour build rapide (dotnet restore + build)
  - o README.md avec documentation

2/ Code principal :

- Créer une classe Commands dans /Commands/Plugin.cs
- Gestion des erreurs (try/catch) et messages utilisateur clairs
- Utiliser Autodesk.AutoCAD.ApplicationServices.Editor pour les interactions utilisateur

3/ Compilation :

- Restaurer les dépendances : dotnet restore
- Compiler : dotnet build -c Release -p:Platform=x64
- Résultat attendu : build réussi, DLL dans bin\x64\Release\net8.0-windows\Plugin.dll
- Aucune exécution de Civil 3D ne doit être lancée automatiquement.

4/ Objectif final :

- Build totalement automatique
- Fichiers structurés proprement pour tests et évolutions
- Cursor doit corriger toutes les erreurs de build jusqu'à obtention d'un ""Build succeeded""
- Aucun lancement externe de Civil 3D n'est autorisé
- Utilisation exclusive de dotnet CLI (pas de MSBuild.exe)
- Chaque projet est autonome et contient toutes ses références dans son .csproj

DEMANDE :

1/ Contexte :

Tu es un développeur C# expert en plugins AutoCAD / Civil 3D. Tu dois concevoir un plugin Civil 3D fiable et robuste, destiné à un usage métier, en respectant les contraintes de l'API Autodesk et les bonnes pratiques .NET.

2/ Objectif :

Développer un plugin Civil 3D permettant de créer une polyligne 2D à partir de points définis par l'utilisateur, soit par saisie manuelle de coordonnées, soit par sélection graphique dans le dessin.

3/ Entrées :

Mode de saisie choisi par l'utilisateur :

- Saisie manuelle : coordonnées (X, Y) entrées dans la ligne de commande.
- Sélection graphique : points sélectionnés dans le dessin.
- Nombre de points variable (minimum 2).
- Interaction utilisateur via la ligne de commande Civil 3D.

4/ Sorties :

- Une polyligne 2D créée et visible dans le dessin Civil 3D.
- Messages clairs dans la ligne de commande :
  - o Confirmation de création
  - o Nombre de points utilisés
  - o Messages d'erreur explicites si nécessaire

5/ Traitement :

- Lancer la commande Civil 3D dédiée
- Demander à l'utilisateur de choisir le mode de saisie
- Récupérer les points dans l'ordre de saisie ou de sélection
- Vérifier la validité des points
- Créer une polyligne 2D AutoCAD à partir des points fournis
- Ajouter la polyligne dans le ModelSpace du dessin courant
- Gérer proprement les transactions et libérations de ressources.

6/ Contraintes :

- Compatibilité Civil 3D / AutoCAD 2026.
- Utilisation exclusive des API officielles Autodesk
- Aucune dépendance externe.
- Gestion robuste des erreurs utilisateur
- Ne pas créer de géométrie si les conditions minimales ne sont pas remplies.

7/ Critères de validation :

- Le plugin compile sans erreur et se charge correctement dans Civil 3D.
- La commande est disponible et exécutable.
- La polyligne générée correspond exactement aux points fournis.
- Les messages utilisateur sont clairs, cohérents et exploitables.
- Aucun crash ou comportement instable n'est observé.