

LLMs

FOR NETWORKS AUTO-RECONFIGURATION: AN MCP-BASED AGENTIC APPROACH*

Abellache Rabah, SAHOUЛИ Sabrina

Master Computer and Networked Systems—Networked Systems

Univ. Évry Paris-Saclay

rabah.abellache@etud.univ-evry.fr

sabrina.sahouli@etud.univ-evry.fr

ABSTRACT

Les réseaux de nouvelle génération, basés sur des technologies telles que la 5G, le Cloud ou l’Edge Computing, ont vu leur complexité croître au point qu’une configuration manuelle ou statique n’est plus envisageable. Si les solutions d’automatisation vont des scripts aux architectures SDN et apportent une certaine souplesse, elles manquent toutes d’une forme d’intelligence contextuelle. L’avènement des modèles de langage de grande taille (Large Language Models) tels que PreConfig, GeNet ou LLM-NetCFG permet d’entrevoir des scénarios où la conception, la traduction et l’adaptation de configurations de réseaux seraient automatiques à partir d’instructions en langage naturel, et où les réseaux pourraient être considérés comme intelligents. Cependant, il subsiste encore beaucoup de travail avant d’aboutir à de tels scénarios. Parmi les limitations : une génération d’erreurs, un manque d’horizon, une dépendance à des datasets propriétaires, une difficulté à s’insérer dans des environnements hétérogènes, etc. Ce mémoire propose de faire interagir un LLM de configuration réseau avec le Model Context Protocol (MCP), un standard d’orchestration récent, et d’examiner dans quelle mesure cela permettrait de construire une architecture d’autoconfiguration et de supervision plus complète et plus interopérable pour les réseaux du futur.

*Code and demonstration available at <https://github.com/Abellache-Rabah/ai-network-engineer/>

CONTENTS

1	Introduction	3
2	Background	4
2.1	Réseau	4
2.2	Configuration	5
2.3	Déploiement	5
2.4	Automatisation	5
2.5	Auto-configuration	5
2.6	Agent	5
3	Approches Traditionnelles (Déterministes)	5
3.1	Méthode CLI (Command Line Interface)	5
3.2	Scripts d'automatisation	5
3.3	Outils d'automatisation -Infrastructure as code (IaC)	6
3.4	Software-Defined Networking - SDN	6
4	l'IA pour la configuration des réseaux	7
4.1	IA en général pour la configuration réseau	7
4.1.1	Intent Based Networking - IBN	7
4.1.2	LUMI comme application concrète de l'IBN	8
4.2	Les Familles de Modèles LLM pour le Réseau	9
4.2.1	Approche 1 : La Traduction d'Intention (Text-to-Config)	9
4.2.2	Approche 2 : L'Assistance Multimodale (Co-Pilote)	10
4.2.3	Approche 3 : L'Automatisation Zero-Touch	12
4.3	Synthèse comparative et identification du gap	14
5	PROBLEMATIQUE ET OBJECTIFS	14
5.1	Positionnement du problème	14
5.2	Objectifs	14
6	Méthodologie / Architecture proposée	15
6.1	Vue d'ensemble de l'architecture	15
6.2	Le protocol MCP	15
6.3	WorkFlow	16
7	Implémentation	16
7.1	Architecture globale du système	16
7.2	Description des composants : Serveurs MCP	16
7.3	Ressources partagées et intégration - SHARED/	18

7.4	Stack technique	18
7.5	Test d’implémentation et défis rencontrés	18
7.5.1	Procédure de configuration	18
7.5.2	Test 1 — Modification d’adresse IP sur une interface existante	19
7.5.3	Test 2 — Migration d’adresse IP vers un nouveau sous-réseau	21
7.5.4	Test 3 - Mise en place du routage dynamique RIP	23
7.6	Conclusion de l’Implémentation	25
7.7	Perspectives et Travaux Futurs	25
8	Conclusion Générale	26

1 INTRODUCTION

L’évolution rapide des infrastructures de communication, marquée par l’avènement des réseaux de nouvelle génération tels que la 5G, le Cloud, et l’Edge Computing, a engendré une complexité sans précédent dans leur gestion (1). Ces environnements modernes, caractérisés par leur nature hétérogène et dynamique, ont rendu la configuration manuelle ou statique des systèmes obsolète (1). Les méthodes conventionnelles sont non seulement sujettes aux erreurs et chronophages, mais elles manquent surtout de l’agilité et de l’adaptabilité requises pour gérer les changements en temps réel dans ces infrastructures complexes (1; 2).

Face à ce défi, l’automatisation, qu’elle soit basée sur des scripts ou sur des outils (Ansible, Puppet, etc.), a offert une flexibilité limitée. Ces solutions ne parviennent pas à interagir de manière simple et générale.

La réponse à cette complexité s’est matérialisée dans le paradigme de l’Intent-Based Networking (IBN). L’IBN représente une évolution majeure vers des réseaux auto-gérés (1; 3). Il permet aux opérateurs d’exprimer des objectifs de haut niveau ou des résultats souhaités (intents) en langage humain, au lieu de spécifier des configurations techniques détaillées (3). L’IBN vise ainsi à réduire la complexité opérationnelle et à minimiser l’intervention humaine (3).

L’Intelligence Artificielle est apparue comme une force de transformation essentielle, visant à automatiser et améliorer les opérations réseau. L’IA, qui inclut des méthodologies avancées comme l’apprentissage automatique (ML) et l’apprentissage profond, permet des analyses et une maintenance prédictives, la détection des anomalies, et l’optimisation des performances. L’intégration de l’Intelligence Artificielle, notamment l’IA générative, a abouti à l’émergence des Grands Modèles de Langage (LLMs), qui sont considérés comme une technologie transformative (1).

L’émergence des modèles de langage de grande taille (Large Language Models ou LLMs), à l’instar de familles de modèles comme GPT, BERT, et LLaMA (1), représente un changement de paradigme pour les opérations et la gestion de réseau (1; 4). Les LLMs permettent d’envisager des scénarios où la gestion de réseau deviendrait autonome et basée sur l’intention, permettant aux ingénieurs de définir des objectifs de haut niveau en langage naturel, que le système traduit ensuite en configurations (1; 5).

1. Des travaux spécialisés ont d’ores et déjà démontré ce potentiel :

- **PreConfig** : a modélisé les tâches d’automatisation de configuration réseau (génération, traduction, analyse) (6)
- **GeNet** : est proposé comme un co-pilote multimodal capable d’interpréter des images de topologie de réseau et de les mettre à jour, en plus de modifier les configurations en fonction de l’intention de l’utilisateur (7)
- **LLM-NetCFG** : vise spécifiquement à réaliser l’auto-configuration pour le Zero-touch Network & Service Management (ZSM), en convertissant les intentions de haut niveau en configurations exécutables (8)

Malgré ces progrès, des limitations subsistent qui freinent l'adoption à grande échelle des LLMs pour l'autoconfiguration réseau :

1. **Génération d'erreurs et fiabilité limitée** : Les LLMs sont sujets aux "hallucinations", pouvant générer des configurations avec des erreurs syntaxiques et sémantiques (6).
2. **Manque de connaissance contextuelle et dépendance aux données** : les LLMs nécessitent des informations contextuelles détaillées et à jour pour générer des sorties précises (9).
3. **Interopérabilité et hétérogénéité** : Les réseaux utilisent une multitude d'outils et de plate-formes ; cependant, il manque une approche commune et structurée pour permettre aux LLMs d'interagir avec cet ensemble hétérogène.

Pour réaliser la promesse des réseaux autonomes, les LLM doivent passer du rôle d'outils passifs de génération de texte et de traduction à celui d'agents autonomes (Agentic AI). Nous proposons une solution qui utilise le protocole MCP, un standard d'orchestration récent, et d'examiner dans quelle mesure cela permettrait de construire une architecture d'autoconfiguration et de supervision plus complète et plus interopérable pour les réseaux du futur. L'objectif est de fournir la couche d'orchestration sécurisée et consciente du contexte nécessaire pour transformer un LLM en un agent.

2 BACKGROUND

dans le background , definir la configuration , automatisation , reseau , deploiement , l'auto-config.

2.1 RÉSEAU

"A network consists of two or more computers that are linked in order to share resources (such as printers and CDs), exchange files, or allow electronic communications. The computers on a network may be linked through cables, telephone lines, radio waves, satellites, or infrared light beams." (24)

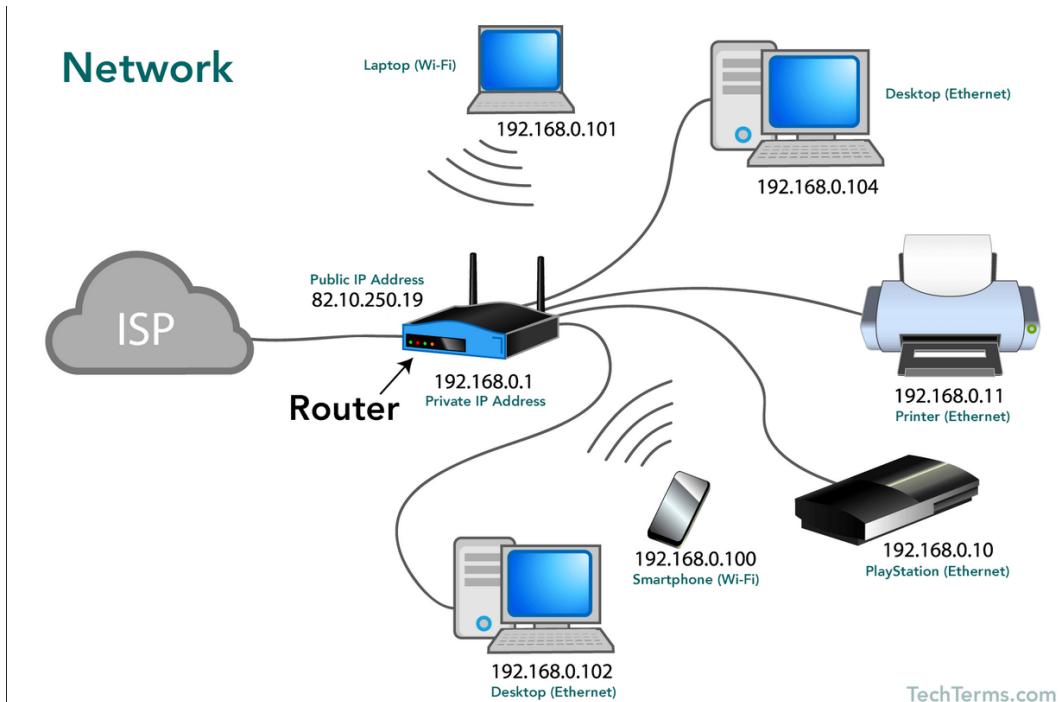


Figure 1: Network topologie

2.2 CONFIGURATION

“Network configuration is the process of assigning network settings, policies, flows, and controls” (22)

par exemple: connect point A to point B.

2.3 DÉPLOIEMENT

“Deployment configurations define the decision operations, target servers, and a version policy for the deployed RuleApps and rulesets. You use deployment configurations to deploy decision services.” (23)

2.4 AUTOMATISATION

“Automation is the application of technology, programs, robotics or processes to achieve outcomes with minimal human input.” (24)

2.5 AUTO-CONFIGURATION

” Self-configuration, the ability to configure network parameters without human intervention. ” (8)

2.6 AGENT

”Un agent IA est un logiciel qui peut interagir avec son environnement, collecter des données et les utiliser pour effectuer des tâches complexes afin d’atteindre des objectifs prédéterminés.” (?)

3 APPROCHES TRADITIONNELLES (DÉTERMINISTES)

Dans cette section, nous allons passer en revue les différentes approches traditionnelle de configuration des réseaux en commençant par : (1) La méthode CLI, (2) Les scripts d’automatisation, (3) Les Outils d’automatisation, (4) Software-Defined Networking

3.1 MÉTHODE CLI (COMMAND LINE INTERFACE)

La configuration manuelle via l’interface en ligne de commande (CLI manuelle) représente la première phase de l’évolution de la configuration reseaux (1) À ce stade, les opérations réseau étaient gérées manuellement par des administrateurs, configurant des équipements tels que des routeurs, switches et des pare-feu . Cependant, cette approche traditionnelle présentait plusieurs incovénients majeurs:

- **Source d’erreurs** : La configuration manuelle par CLI est considérée comme la principale source d’erreurs dans les réseaux.
- **Complexité et Inefficacité** : Les réseaux moderne (5G, Cloud, Edge) sont devenus trop complexes pour être configurés manuellement de maniere statique, La gestion réseau traditionnelle impliquait souvent l’utilisation de multiples interfaces de gestion qui intégraient des commandes CLI complexes, Les tâches manuelles gaspillent du temps et engendrent des coûts
- **Absence de standardisation** : Le langage de configuration réseau est très technique, composé de formats non standardisés, de symboles et de terminologies spécifiques au domaine.

(10).

3.2 SCRIPTS D’AUTOMATISATION

Avec la reconnaissance des limites des configurations manuelles, la nécessité d’automatisation a conduit à la phase des scripts statiques. Ces scripts constituent la deuxième phase de l’évolution

de la configuration réseau. Les scripts d'automatisation sont développés en utilisant des langages de programmation ou de script classiques tels que Python, Perl, ou Shell(Bash). Ces scripts visent à standardiser et répéter les séquences de commandes. Ils permettent d'appliquer une configuration spécifique à un ensemble d'appareils de manière plus rapide et cohérente qu'une intervention humaine directe.(15)(16) Une fois testé et validé, un script assure que la même série d'actions est exécutée de la même manière à chaque fois, garantissant ainsi une meilleure fiabilité et une réduction des erreurs humaines par rapport à la CLI manuelle.(16) Cependant, cette méthode présente ses propres limites:

- **Manque d'adaptabilité:** C'est le principal inconvénient des scripts statiques. Ils exécutent une série d'actions prédefinies sans tenir compte de l'état actuel ou des conditions changeantes du réseau.(17)
- **Maintenance lourde:** Dès que l'infrastructure change, le script doit être modifié manuellement, ce qui rend la maintenance et l'évolutivité difficile et coûteuse en temps.(17)

3.3 OUTILS D'AUTOMATISATION - INFRASTRUCTURE AS CODE (IAC)

Cette étape a vu l'émergence d'outils d'automatisation et de capacité de scripting, réduisant la nécessité d'intervention manuelles. Des outils de gestion de configuration comme Ansible, Puppet .. ont permis aux administrateurs réseau d'automatiser la configuration des appareils à grande échelle (1). Les outils d'automatisation sont caractérisés par la centralisation et l'idempotence, cette dernière garantit le même résultat après plusieurs exécutions. Les tâches répétitives sont également exécutées rapidement, ce qui contribue à la réduction des erreurs manuelles.

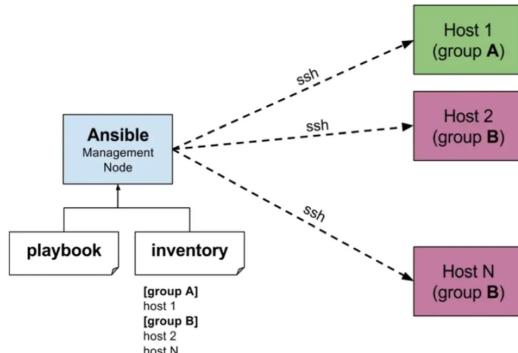


Figure 2: Sample figure caption.

Malgré cette évolution, cette phase présentait des limitations majeures:

- **Complexité croissante :** La gestion continuait de souffrir de la complexité des environnements multi-fournisseurs et hybrides(on-premise + cloud)
- **Manque d'intelligence :** Bien qu'ils aient introduit des scripts d'automatisation, ceux-ci restent statiques, et ne pouvant pas s'adapter aux conditions dynamiques et changeantes du réseau.
- **Supervision humaine :** L'expertise humaine restait nécessaire pour le diagnostic des pannes complexes

3.4 SOFTWARE-DEFINED NETWORKING - SDN

La phase des réseaux programmables a transformé la gestion des réseaux en remédiant aux limitations des scripts statiques qui ne pouvaient pas s'adapter aux conditions changeantes du réseau (1). La transition vers le SDN a été motivée par le besoin de programmabilité et de contrôle centralisé. Le SDN sépare le plan de contrôle (control plan) du plan de données (data plan). Il utilise des protocoles tels qu'OpenFlow et des contrôleurs SDN centralisés comme ONOS et OpenDaylight pour assurer

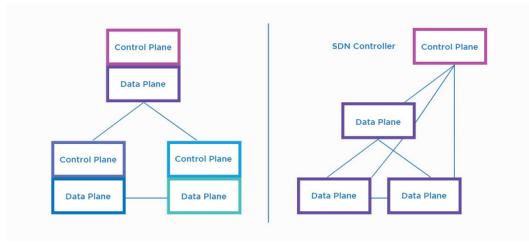


Figure 3: SDN vs Traditional Networking (11)

la programmabilité et le contrôle centralisé ce qui permet une gestion plus flexible des ressources réseau. (1)

Malgré ses avancées, l'approche SDN n'a pas été suffisante pour répondre aux besoins d'autonomie complète des réseaux modernes:

- **Problème de scalabilité** : le contrôleur centralisé peut devenir un goulet d'étranglement dans les grands réseaux. (12)
- **Manque d'intelligence**: Le SDN offre la programmabilité, mais manque d'intelligence décisionnelle pour permettre aux réseaux de se configurer et de se réparer eux-même. (12)

Malgré l'avancée qu'il représente, l'IBN de première génération n'a pas atteint le niveau d'autonomie complète souhaité par les réseaux modernes:

- **Dépendance aux règles statiques**: La traduction des intentions reposait sur des moteurs de règles statiques et des modèles (templates) prédéfinis (3). Ces systèmes manquaient de flexibilité pour s'adapter rapidement aux conditions réseau complexes ou aux exigences qui n'avaient pas été prévues lors de la conception des modèles
- **Complexité de l'expression et traduction ambiguë**: La qualité de la traduction d'une intention dépendait fortement de la capacité de l'utilisateur à l'exprimer précisément (3).
- **Manque d'intelligence adaptative**: Bien que l'IBN ait jeté les bases des réseaux auto-gérés, il n'avait pas encore atteint les niveaux d'adaptabilité et de généralisation nécessaires pour une gestion entièrement autonome

Ces limitations ont souligné la nécessité de systèmes plus intelligents pour gérer l'incertitude et la complexité croissante, ouvrant la voie à l'intégration de l'Intelligence Artificielle et des Grands Modèles de Langage (LLMs) dans les réseaux (1) (2)

4 L'IA POUR LA CONFIGURATION DES RÉSEAUX

4.1 IA EN GÉNÉRAL POUR LA CONFIGURATION RÉSEAU

La première vague d'intelligence artificielle dans les réseaux s'est concentrée sur deux domaines principaux :

4.1.1 INTENT BASED NETWORKING - IBN

L'approche Intent-Based Networking (IBN) est un paradigme qui est apparu pour répondre à la complexité croissante des réseaux et au besoin d'automatisation des configurations (1) (3). L'IBN se concentre sur le résultat souhaité (le "quoi"), plutôt que sur les étapes de configuration spécifiques (le "comment") (3) (13). Selon l'Internet Engineering Task Force (IETF), une intention est définie comme : "A set of operational goals (that a network should meet) and outcomes (that a network is supposed to deliver), defined in a declarative manner without specifying how to achieve or implement them" (14).

Le fonctionnement de l'IBN s'articule autour d'une boucle de contrôle fermée (Closed Loop Automation)(13) (3), composée de cinq composants principaux :

- **Intent Profiling (Expression)** : L'utilisateur exprime une intention de haut niveau à l'aide d'interfaces graphiques (GUI) ou de langages spécifiques au domaine (langages IBN), comme Nile.
- **Intent Translation**: L'intention abstraite est traduite en politiques réseau de bas niveau (13) (3) a l'aide d'un moteur de règles statique et l'utilisation de modèles prédéfinis (blueprints).
- **Intent Resolution**: Cette étape vérifie si l'intention est réalisable et si elle n'entre pas en conflit avec les intentions déjà déployées (3)
- **Intent Activation**: Une fois résolue, l'intention est déployée sur l'infrastructure physique et virtuelle en réservant les ressources nécessaires, souvent via des systèmes comme SDN (3)
- **Intent Assurance**: Ce composant surveille le réseau en continu pour garantir que les performances restent conformes à l'intention exprimée tout au long de son cycle de vie (3)

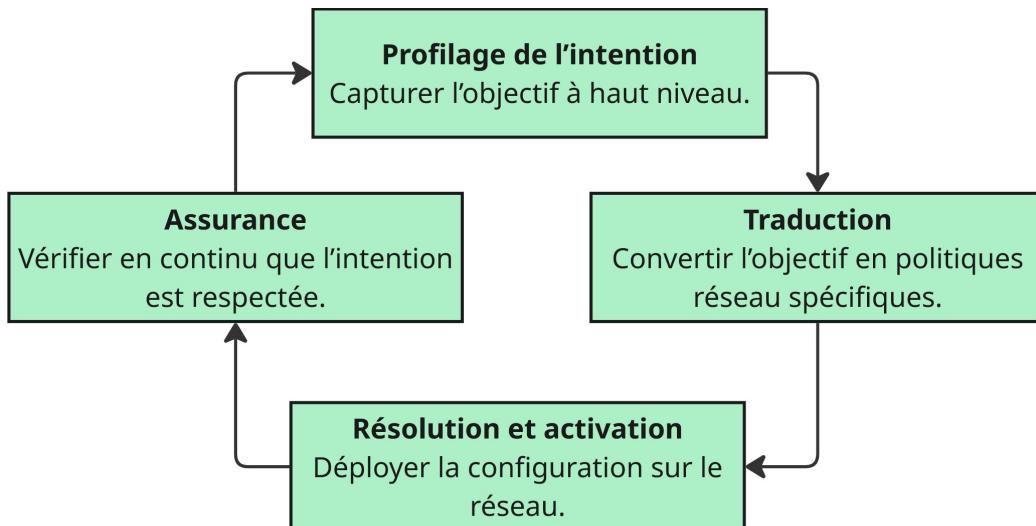


Figure 4: composants d'un système IBN

4.1.2 LUMI COMME APPLICATION CONCRÈTE DE L'IBN

: Le système LUMI (Linguistically Understood Management Interface) est un exemple de système IBN qui utilise les techniques de NLP et de ML pour traduire le langage naturel en configurations (18). Le pipeline de LUMI est composé de quatre modules:

- **Extraction d'Information**: Utilise le ML (par exemple, les modèles Bi-LSTM) pour effectuer la Reconnaissance d'Entités Nommées (NER) afin d'extraire et d'étiqueter les informations pertinentes de l'intention en langage naturel (18).
- **Assemblage d'Intention**: Construit une intention structurée et syntaxiquement correcte en utilisant un langage d'abstraction intermédiaire appelé Nile (Network Intent LanguagE). Nile permet d'exprimer des politiques de haut niveau, telles que les ACL (Access Control List), le QoS (Quality of Service).. (18).
- **Confirmation d'Intention**: L'intention Nile est présentée à l'opérateur pour confirmation et feedback, permettant au système d'apprendre continuellement et de réduire les ambiguïtés inhérentes au langage naturel (18).
- **Déploiement d'Intention**: Compile l'intention Nile confirmée en commandes de configuration exécutables (comme dans le langage Merlin) pour les déployer sur les équipements réseau (18).

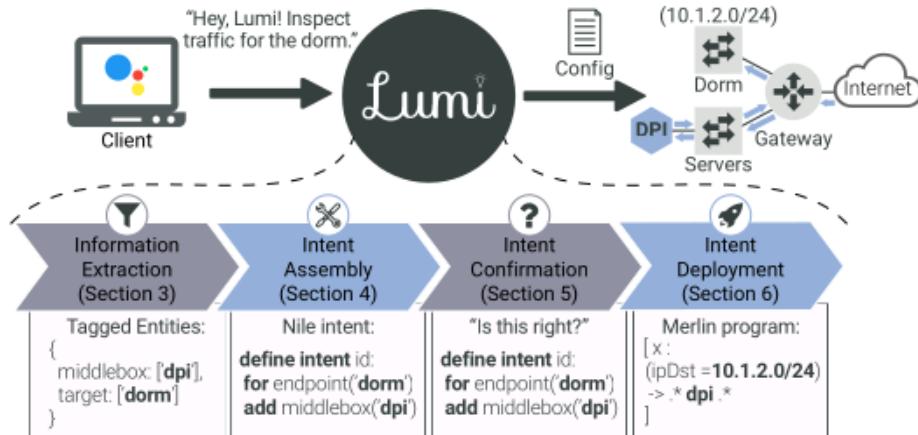


Figure 5: The four modules of LUMI
(18)

4.2 LES FAMILLES DE MODÈLES LLM POUR LE RÉSEAU

L’arrivée des LLM a permis de traduire le langage naturel (Intentions) en commandes techniques. Nous identifions trois générations d’approches LLM dans la littérature récente.

4.2.1 APPROCHE 1 : LA TRADUCTION D’INTENTION (TEXT-TO-CONFIG)

PreConfig

PreConfig est un modèle de langage pré-entraîné conçu spécifiquement pour l’automatisation de la configuration réseau (NCA). Il est conçu pour aborder la complexité et la variété des tâches NCA en les formulant comme des problèmes de transformation texte-à-texte (6). L’approche de PreConfig se déroule en deux étapes principales:

- **Pré-entraînement adaptatif à la tâche (Stage 1):** Il construit un corpus spécialisé en extrayant des données des manuels de fournisseurs (tels que Cisco et Juniper) et des forums communautaires. Il utilise le modèle PLBART (406 millions de paramètres) comme modèle de base, pré-entraîné sur du code et du langage naturel, et injecte le corpus de configuration via l’apprentissage par transfert pour renforcer sa compréhension du domaine (?).
- **Fine-tuning multi-tâches (Stage 2):** Il affine le modèle sur les tâches en aval (génération, traduction, analyse) en utilisant un framework d’apprentissage multi-tâches. Pour obtenir des données de supervision, il utilise une technique de data mining qui implique l’interaction d’un agent LLM avec un environnement simulé, avec une validation par des outils d’analyse de configuration (comme Batfish) pour garantir l’exactitude des données (?).

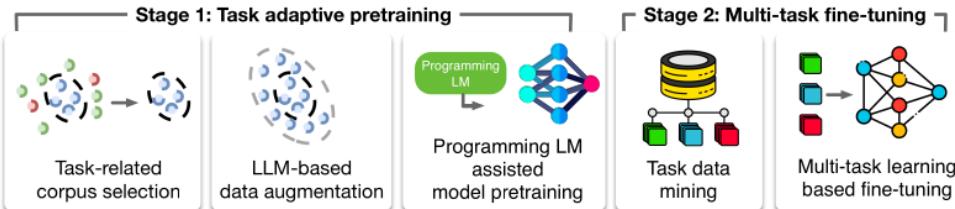


Figure 6: Overall framework of PreConfig

Les tâches principales de PreConfig sont la génération, la traduction et l'analyse des configurations

Scénarios de Test et Cas d'Usage

PreConfig est évalué sur des tâches fondamentales d'automatisation de la configuration réseau (NCA) (6)

- **Génération de Configuration :** Transformer des exigences de haut niveau (exprimées en langage naturel) en configurations réseau exécutables. Par exemple, la génération de configurations pour les protocoles établis tels que RIP, OSPF et BGP (6).
- **Traduction de Configuration:** Convertir la syntaxe de configuration d'un fournisseur à celle d'un autre (par exemple, Cisco à Juniper) (6).
- **Analyse de Configuration:** Transformer la configuration d'un appareil en texte en langage naturel (intentions) (6).

Lors d'une évaluation comparative, PreConfig a démontré une amélioration de 14,69 en score BLEU et de 2,59 en ROUGE par rapport à GPT-4 pour la tâche de génération de configuration (6).

Les Limites du Preconfig

- **Rareté des données de supervision et du corpus spécialisé :** Obtenir des corpus de haute qualité et des données de supervision est un défi majeur dans le domaine de la configuration réseau, en partie à cause des exigences de sécurité. Pour la tâche de traduction de configuration, les données de supervision sont même considérées comme presque inexistantes (6).
- **Erreurs syntaxiques et sémantiques malgré l'adaptation:** Bien que PreConfig surpassé les modèles génériques comme ChatGPT et GPT-4, ces derniers peuvent encore produire des erreurs de syntaxe et de sémantique dans des configurations spécifiques (comme pour community-list et route-map), soulignant le défi inhérent à la génération de configuration réseau précise par les LLM (6).
- **Coût de l'intégration des connaissances professionnelles:** L'intégration des connaissances professionnelles robustes dans les modèles est difficile. PreConfig doit passer par deux étapes d'adaptation (pré-entraînement adaptatif à la tâche et fine-tuning multi-tâches) pour injecter le corpus de configuration provenant des manuels de fournisseurs (Cisco et Juniper) et des forums communautaires.(6)
- **fossé de l'exécution:** Il génère du texte, mais ne possède aucun moyen de se connecter au réseau pour appliquer la configuration ou vérifier si elle est valide dans le contexte actuel. Il agit comme un traducteur aveugle, sans conscience de l'état réel de l'infrastructure.

4.2.2 APPROCHE 2 : L'ASSISTANCE MULTIMODALE (CO-PILOTE)

GeNet

GeNet est un copilote multimodal conçu pour aider les ingénieurs réseau dans la conception de topologies et la configuration de dispositifs (7). Il vise à fournir une assistance conviviale et holistique aux ingénieurs réseau, en exploitant les capacités d'IBN et des LLMs multimodaux. Il permet d'utiliser des croquis de topologie et des images comme spécifications réseau directes (7). Les principaux composants architecturaux de GeNet sont :

1. **Entrées du Système:** Il reçoit les spécifications du réseau, qui comprennent deux types d'informations :
 - **L'intention Utilisateur:** exprimée en langage naturel .
 - **Les Spécifications d'Entrée:** qui incluent l'image de la topologie et les configurations des appareils (7).
2. **Module principaux** L'architecture de GeNet est divisée en plusieurs modules clés, utilisant notamment le modèle GPT-4 pour ses capacités avancées.
 - **module de Compréhension de topologie (Topology Understanding Module):** il est responsable de l'interprétation des données visuelles, telles que les images de topo-

gie (y compris les croquis faits à la main et de faible qualité) (7), il utilise GPT-4 Vision, il génère une représentation textuelle de la topologie.

- **Module d'Implémentation de l'Intention (Intent Implementation Module):** Ce module est chargé d'appliquer l'intention de l'utilisateur en mettant à jour les spécifications réseau. Il utilise L'assistant API d'OpenAI pour créer un assistant d'IA personnalisé pour effectuer des tâches complexes. Le module met à jour la représentation textuelle de la topologie, configure les nouveaux composants ajoutés et modifie les configurations d'appareils existantes si nécessaire(7).

Le cadre GeNet inclut également un Cadre d'Auto-Évaluation (Auto Evaluation Framework) pour valider les solutions générées.

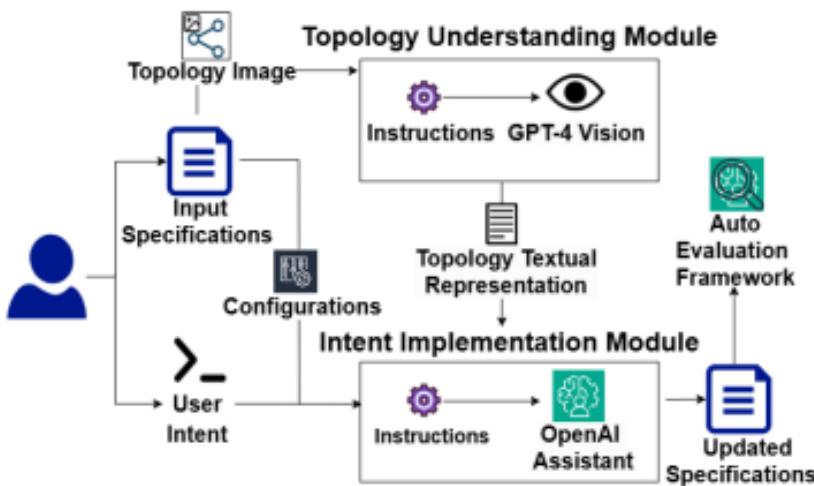


Figure 7: An overview of the GeNet framework
(7)

Scénarios de Test et Cas d'Usage

GeNet a été évalué dans des scénarios réalistes d'Ingénierie Réseau Basée sur l'Intention (IBN), s'inspirant d'exercices de certification Cisco et d'entretiens avec des experts IT.

”We conduct a comprehensive assessment of GeNet’s ability to handle a variety of intents, by evaluating its performance in a series of scenarios, characterized by varying complexities and input specification quality, formulated based on Cisco certification exercises and interviews with IT experts.” (7)

Les scénarios de test se divisent en deux catégories:

- **Mises à jour de Topologie:** Ajout d'une DMZ, ajout d'un DRA, connectivité Internet, ajout de PCs locaux et ajout de serveurs de communication.
- **Modifications de Configuration:** Exportation du trafic IP, pare-feu de zone de base, accès CLI basé sur les rôles, liste d'accès basée sur le temps et pare-feu transparent IOS.

Une évaluation approfondie de la capacité de GPT-4 à interpréter les images de topologie a été menée en utilisant neuf variantes d'images pour chaque scénario, couvrant différents formats de visualisation (GNS3, PowerPoint, croquis dessinés à la main) et différents types de mise en page (Normal, Sans Étiquettes sur les Liens, et Mise en page Désordonnée (Messy Layout)). GeNet a montré qu'il pouvait gérer efficacement les spécifications réseau de qualité variable, y compris les mises en page désordonnées ou les informations manquantes (7).

Les Limites du GeNet

- **Dépendance au LLM multimodal:** GeNet repose sur les capacités de GPT-4, qui surpassé les autres modèles populaires dans les évaluations de réseau (7).
- **Configurer des équipements à partir de zéro est plus difficile:** Le modèle est nettement meilleur pour modifier des configs existantes que pour créer des configurations à partir de zéro. Dans les scénarios, les nouveaux équipements sont configurés correctement dans seulement 72% des cas, contre 92% lorsque la config existe déjà (7).
- **Les tâches de mise à jour de topologie sont plus difficiles:** Les scénarios impliquant une évolution de l'architecture se révèlent nettement plus complexes que de simples ajustements de configuration. Ils exigent en effet l'ajout de nouveaux équipements, leur configuration complète ainsi que le maintien de la cohérence de l'ensemble du réseau. Dans ces conditions, le modèle montre une performance plus limitée. (7)
- **Dépendance Humaine (Manque d'Autonomie):** GeNet est conçu pour assister, pas pour gérer le réseau de manière autonome en temps réel.

4.2.3 APPROCHE 3 : L'AUTOMATISATION ZERO-TOUCH

LLM-NetCFG

LLM-NetCFG (Large Language Model-Network Configuration) est un framework visant à réaliser l'auto-configuration réseau sans intervention humaine (zero-touch) en utilisant un LLM. Il est conçu pour traduire les intentions exprimées en langage naturel en configurations réseau complètes, qu'il génère, vérifie et déploie automatiquement (8). L'architecture du système repose sur trois modules principaux qui coordonnent un flux de travail cyclique:

- **Module LLM:** Déploie et exploite un LLM local (par exemple, Zephyr-7B , un modèle basé sur Mistral) pour traiter les intentions et générer les configurations (8).
- **Module Vérificateur (Verifier):** Reçoit la configuration générée et vérifie les erreurs de syntaxe et l'applicabilité/correction, notamment en utilisant Batfish pour la vérification syntaxique (8).
- **Module Orchestrator (Orchestrator) :** Coordonne le flux entre les modules LLM et Vérificateur, construit des prompts personnalisés (Classification, Traduction, Génération) et gère le cycle de vérification et de rétroaction vers le LLM en cas d'erreur (8).

Les avantages principaux de LLM-NetCFG incluent l'exécution locale (confidentialité), l'autonomie (auto-configuration, auto-réparation, auto-optimisation) et la vérification intégrée (génération, validation, correction). Le framework est agnostique au modèle et modulaire

"In particular, LLM-NetCFG is model-agnostic and modular, providing flexibility to integrate other management functions and facilitate its enhancement by specialized modules oriented to fine-tuning." (8)

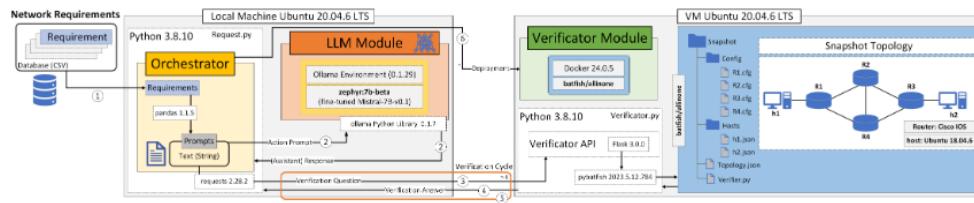


Figure 8: LLM-NetCFG Structure
(8)

Scénarios de Test et Cas d'Usage LLM-NetCFG a pour objectif de générer et vérifier des configurations réseau pour le Zero-touch Service Management (ZSM) (8).

- **Tâches de Configuration:** Le système a été testé sur des tâches courantes telles que la création et la suppression de connexions, l'établissement de listes de contrôle d'accès

(ACL), la configuration des règles de routage OSPF et la création de communications sécurisées à l'aide de technologies de tunneling (TN) (8).

- **Environnement de Prototype:** Un prototype a été utilisé sur un réseau de topologie en maillage partiel comprenant quatre routeurs Cisco, avec deux utilisateurs finaux connectés (8).
- **Évaluation :** Les expériences ont utilisé un jeu de données de 90 intentions de configuration réparties en quatre types (CP pour les propriétés d'interface, ACL pour les listes de contrôle d'accès, RP pour le routage et TN pour les configurations de tunnels). Le système a correctement classifié 92,2 % des exigences, démontrant sa capacité à comprendre les intentions (8).

Les Limites du LLM-NetCFG

- **Hallucinations par manque de données réelles :** Bien que le système valide la syntaxe, le LLM n'a pas d'accès direct aux données opérationnelles du réseau en temps réel (ex : tables ARP, état des liens, voisins CDP). En conséquence, il "hallucine" souvent des paramètres contextuels (comme inventer une adresse IP libre ou cibler une interface qui n'existe plus physiquement), rendant la configuration inapplicable en production.(8).
- **Risque de Boucle Infinie:** Un seuil est nécessaire dans le cycle de vérification pour éviter une boucle infinie si le modèle génère continuellement une configuration non approuvée par le Vérificateur (8).
- **Hétérogénéité :** repose sur des scripts spécifiques ("glue code") codés en dur pour connecter le LLM à des outils précis (comme Batfish). Cette approche **Ad-Hoc** n'est pas scalable.

Table 1: Comparaison minimaliste et vérifiée des architectures LLM pour l'automatisation réseau

Approche / Modèle	Config	Vérif. / Valida-tion	Déploi. / Exécut.	Contexte / Temps Réel	Agentique	Catégorie Principale
I. Traduction d'Intention (Text-to-Config)						
PreConfig	✓	✗	✗	✗	✗	Text-to-Config
NETBUDDY	✓	✓	✗	✗	✓	Text-to-Config
LUMI (IBN)	✓	✓	✓	✗	✓	Text-to-Config
II. Co-Pilot et Assistance						
GeNet	✓	✓	✗	✓	✓	Co-Pilot
III. Zero-Touch (Autoconfiguration)						
LLM-NetCFG	✓	✓	✓	✗	✓	Zero-Touch
IV. Solution Proposée (Agentique Modulaire)						
LLM + MCP	✓	✓	✓	✓	✓	Agent

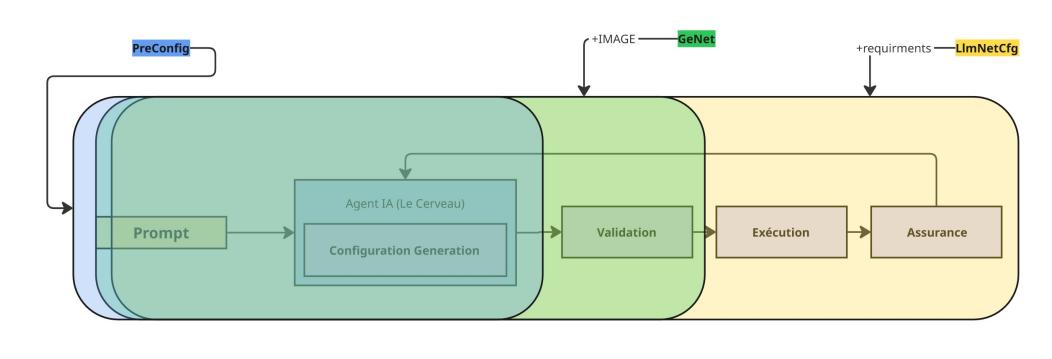


Figure 9: Comparaison schématique des architectures LLM

4.3 SYNTHÈSE COMPARATIVE ET IDENTIFICATION DU GAP

Le tableau ci-dessous résume notre analyse par familles. Il met en évidence qu'aucune approche actuelle ne combine autonomie, contexte temps réel et interopérabilité.

Conclusion de l'état de l'art Aucune des solutions actuelles ne résout le problème de l'hétérogénéité des outils de manière scalable. Elles reposent soit sur des corpus statiques, soit sur des scripts d'intégration ad-hoc (8). Il manque une couche d'abstraction standardisée qui permettrait au LLM de découvrir et d'utiliser n'importe quel outil réseau (scanner, configurateur, vérificateur ...) sans re-développer de modèle. C'est ce gap que notre approche vis à combler.

5 PROBLEMATIQUE ET OBJECTIFS

5.1 POSITIONNEMENT DU PROBLÈME

malgré des avancées notables démontrées par des frameworks comme **PreConfig**, **GeNet**, **LLM-NetCfg**, l'utilisation des LLM dans les environnements de production reste limitée par plusieurs obstacles technologiques majeurs :

1. **Le fossé de l'exécution (The Execution Gap):** Les LLMs sont des moteurs de génération de texte / code (1) (6). Bien qu'ils puissent produire des configurations et scripts valides, ils n'ont pas la capacité native d'interagir avec les équipements physiques pour appliquer ces configurations, déployer le code, ou récupérer des données d'état d'infrastructure en temps réel.
2. **Hallucination:** Comme constaté avec *PreConfig*, un LLM peut produire une syntaxe correct mais erronée s'il n'a pas le contexte exact et à jour du réseau (topologie, les adresses, état des interfaces). Sans données réelles, le modèle hallucine une configuration basée sur des données obsolètes plutôt que sur la réalité du terrain.
3. **Hétérogénéité:** Les réseaux modernes sont hétérogènes. Pour qu'un LLM se comporte comme un véritable administrateur autonome, il doit pouvoir interagir avec une variété d'outils (Serveurs, APIs, Contrôleurs, outils de monitoring) y compris les nouveaux outils qui sont apparus. Coder des intégrations spécifiques pour chaque outil n'est pas scalable et difficile à maintenir.

La problématique centrale de ce mémoire peut donc se formuler ainsi :

Comment transformer un LLM, outil passif de génération de texte, en un agent réseau autonome et fiable, capable d'interagir de manière standardisée avec un environnement hétérogène pour réaliser des tâches d'auto-configuration

5.2 OBJECTIFS

Pour répondre à cette problématique, ce travail de recherche propose une nouvelle approche architecturale basée sur le paradigme de l'**Agentic AI** et l'utilisation du standard emergent Model Context Protocol.

Les objectifs principaux sont :

1. **architecture agentique modulaire:** Proposer une architecture où le LLM n'est plus isolé, mais agit comme un cerveau capable d'appeler des outils externes différents.
2. **Contexte en temps réel :** Concevoir une architecture qui permet à l'agent LLM de récupérer et d'intégrer le contexte réseau actuel (topologie, statut des interfaces, adresses IP) avant de générer ou de modifier des configurations pour réduire les hallucinations.

Ce travail vise finalement à démontrer que l'adoption d'un standard d'interopérabilité comme MCP est la "pièce manquante" pour passer à des réseaux véritablement autonomes et intelligents.

6 MÉTHODOLOGIE / ARCHITECTURE PROPOSÉE

Cette section détaille l'approche architecturale concue pour surmonter les limitations identifiées dans la section problématique 5.1. Nous proposons une architecture d'IA Agentique (Agentic AI) modulaire, où le LLM n'est plus isolé mais orchestre des outils spécialisés via un protocole standardisé : le **Model Context Protocol MCP**

6.1 VUE D'ENSEMBLE DE L'ARCHITECTURE

Contrairement aux approches monolithiques où le LLM tente de tout faire, notre méthodologie repose sur le découplage entre le Cerveau (LLM) et les Bras (les outils)

L'architecture se divise en 3 couches :

- 1. Couche Cognitive (Agent):** Le LLM qui raisonne, planifie et décide des actions à faire.
- 2. Couche d'Orchestration (MCP Client):** C'est le cœur du système. Le Client MCP agit comme une passerelle sécurisée entre IA et les outils.
- 3. Couche Instrumentale (MCP Servers) :** Les outils spécialisés exposés comme des services.

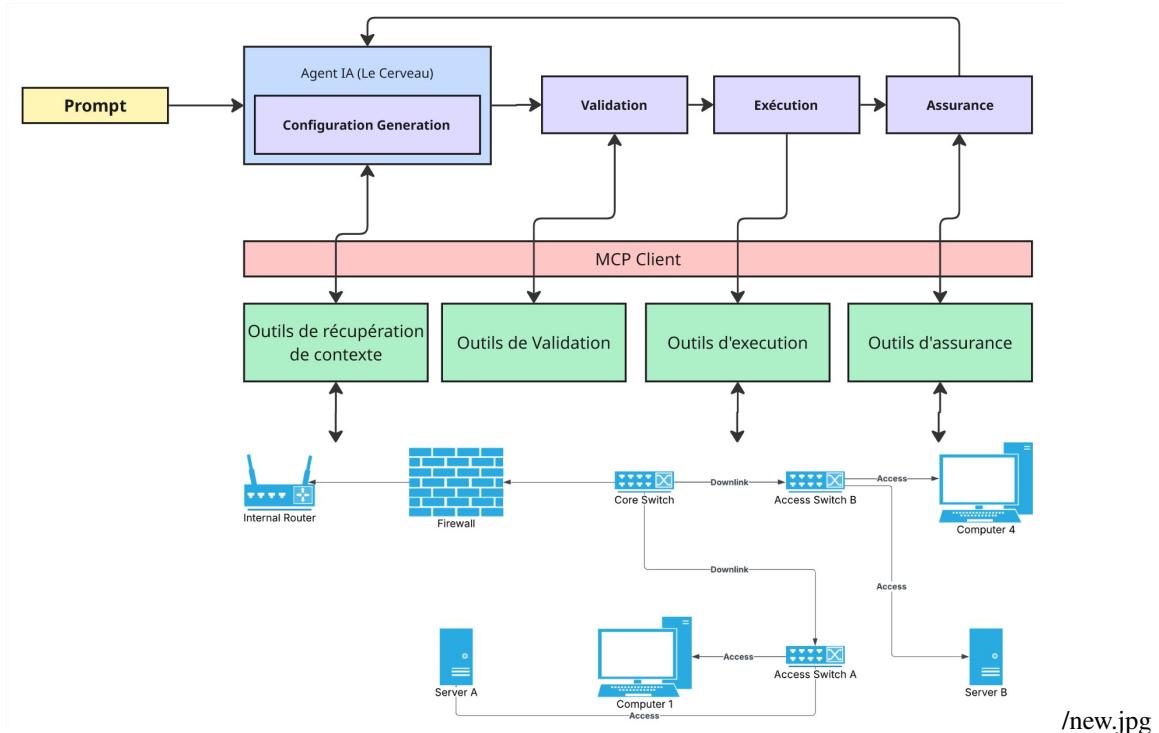


Figure 10: Architecture proposée

6.2 LE PROTOCOL MCP

Au cœur de notre solution architecturale se trouve le Model Context Protocol (MCP). Il s'agit d'un standard ouvert (open standard) introduit par Anthropic en novembre 2024, rapidement adopté par l'industrie (25). Le MCP est la **pièce manquante** qui permet de surmonter le fossé d'exécution (Execution Gap) et les problèmes d'hétérogénéité des outils réseau. Il est souvent comparé à un "port USB-C pour les applications d'IA" (USB-C port for AI applications), car il fournit une méthode standardisée et unifiée pour connecter les modèles d'IA à des systèmes externes (26)

Le rôle principal du MCP est de **transformer le LLM, outil passif de génération de texte/code, en un agent réseau autonome** et fiable capable d’interagir de manière standardisée avec un environnement hétérogène

Dans notre contexte réseau, MCP joue un rôle double :

- **Contexte Injection:** Il permet à l’agent de lire l’état du réseau en temps réel pour réduire les hallucinations (27).
- **Tool Executions:** Il expose les outils différents sous forme de fonctions exécutables par le LLM (26).

6.3 WORKFLOW

Notre agent opère selon une boucle de contrôle fermée et récursive, composée de cinq étapes clés. Ce cycle est conçu pour être entièrement autonome

1. **Récupération du Contexte** : Avant toute génération, l’agent doit construire une image exacte du réseau. Il interroge le **Network state observer** afin d’obtenir la topologie réelle et la **Source of Truth** pour connaître l’état déclaré (Vlans, IPs ...), ce qui minimise drastiquement les risques d’hallucination.
2. **Génération de Configuration** : Fort du contexte et de l’intention exprimée dans le prompt, le LLM génère la configuration cible. Respectent les règles dans *Knowledge Retrieval Module*.
3. **Validation** : Après la génération il va la vérifier avec un vérificateur comme Batfish.
4. **Deployment** : L’agent appelle les outils via MCP pour pousser les configurations.

7 IMPLÉMENTATION

Cette section présente l’implémentation technique de la solution proposée, reposant sur un agent assisté par intelligence artificielle et s’appuyant sur le Model Context Protocol (MCP). L’objectif de cette implémentation est de démontrer la faisabilité d’une orchestration réseau automatisée, contrôlée et vérifiable, en environnement simulé.

7.1 ARCHITECTURE GLOBALE DU SYSTÈME

La solution est conçue selon une architecture micro-services modulaire, dans laquelle chaque fonction clé de l’ingénierie réseau est encapsulée au sein d’un serveur MCP indépendant. Cette approche permet une séparation claire des responsabilités, améliore la maintenabilité du code et facilite l’évolution progressive du système. L’architecture est structurée en trois couches principales :

1. **Couche Agent / Client MCP:** Cette couche correspond à l’interface utilisateur de l’agent (Gemini). Elle est responsable de l’interprétation des intentions formulées en langage naturel et de l’orchestration des appels aux outils exposés par les serveurs MCP.
2. **Couche Serveurs MCP:** Elle regroupe sept serveurs spécialisés, chacun représentant une compétence métier spécifique de l’ingénierie réseau (sécurité, déploiement, supervision, validation, etc.). Les serveurs exposent leurs fonctionnalités sous forme d’outils MCP appelables de manière contrôlée par l’agent.
3. **Couche Infrastructure et Simulation:** Cette couche repose sur un environnement de simulation réseau basé sur GNS3, complété par un ensemble de ressources partagées (shared/) constituant la représentation physique et logique du réseau (inventaire, topologie, utilisateurs).

7.2 DESCRIPTION DES COMPOSANTS : SERVEURS MCP

Nous avons implémenté sept serveurs MCP, chacun assimilable à un rôle spécialisé :

1. Auditor

- **Rôle:** Le serveur Auditor est chargé de garantir la conformité des configurations réseau et d'évaluer les risques de sécurité.
- **Implémentation:** Il compare les configurations des équipements à un ensemble de règles de sécurité prédéfinies , telles que l'obligation de chiffrer les mots de passe ou la désactivation de services non essentiels. Il simule également une analyse de vulnérabilités basée sur les versions logicielles détectées (logique de type CVE)
- **Outils exposés:** check_compliance, scan_vulnerabilities

2. Deployer

- **Rôle:** Le serveur Deployer est le seul composant autorisé à modifier l'état du réseau.
- **Implémentation:** Il s'appuie sur des connexions Telnet vers les équipements simulés GNS3, via la bibliothèque shared.gns3_utils. Avant toute application, il est capable de calculer les différences entre la configuration existante et la configuration cible, permettant un mode dry-run. Un mécanisme de rollback est également prévu afin de limiter l'impact des erreurs de déploiement.
- **Outils exposés:** deploy_config, get_config_diff, rollback

3. IPAM

- **Rôle:** Le serveur IPAM assure la gestion centralisée de l'espace d'adressage afin d'éviter toute allocation arbitraire d'adresses IP.
- **Implémentation:** Il maintient une base de données locale simulée des sous-réseaux connus et des adresses déjà attribuées. Il est capable de calculer le taux d'utilisation d'un sous-réseau et de fournir dynamiquement la prochaine adresse disponible.
- **Outils exposés:** allocate_ip, list_subnet, get_subnet_usage

4. Librarian

- **Rôle:** Le serveur Librarian constitue la Source of Truth (SoT) du système
- **Implémentation:** Il centralise l'accès aux fichiers décrivant l'état logique et physique du réseau (inventory.yaml, topology_physical.yaml). Il implémente également un mécanisme de recherche documentaire de type RAG (Retrieval-Augmented Generation) simulé, fournissant à l'agent des informations techniques contextualisées.
- **Outils exposés:** get_device_info, list_devices, search_docs

5. Observer

- **Rôle:** Le serveur Observer est responsable de la collecte de l'état réel du réseau.
- **Implémentation:** Il se connecte aux équipements pour exécuter des commandes de diagnostic telles que ping ou show ip interface brief.
- **Outils exposés:** check_reachability, detect_link_failures

6. TrafficGen

- **Rôle:** Le serveur TrafficGen permet de valider les changements réseau par des tests fonctionnels.
- **Implémentation:** Il simule le comportement d'outils de génération de trafic (ex. iperf3) afin d'évaluer la connectivité et la bande passante après un déploiement.
- **Outils exposés:** run_traffic_test

7. Verifier

- **Rôle:** Le serveur Verifier vérifie avant toute modification du réseau.
- **Implémentation:** Il s'interface avec l'outil Batfish afin d'effectuer une analyse statique des configurations candidates. Cette analyse permet de détecter des erreurs de syntaxe, des références manquantes ou des incohérences sémantiques avant le déploiement effectif.
- **Outils exposés:** verify_device_config

7.3 RESSOURCES PARTAGÉES ET INTÉGRATION - SHARED/

Le dossier shared/ constitue la couche d'intégration du système. Il fournit un référentiel commun évitant toute duplication ou divergence d'état entre les serveurs.

- **inventory.yaml**: Inventaire logique des équipements (identité, IP de gestion, ports GNS3).
- **topology_physical.yaml**: Description de la topologie physique et des liens entre équipements.
- **gns3_utils.py**: Bibliothèque Python développée pour abstraire les connexions Telnet vers GNS3.

7.4 STACK TECHNIQUE

L'implémentation de la solution repose principalement sur:

- **Python**: Choisi pour sa maturité, son écosystème riche et sa compatibilité avec les outils d'automatisation réseau
- **Model Context Protocol(MCP)**: Assure les interactions entre l'agent IA et les différents composants.
- **GNS3**: L'environnement réseau est simulé à l'aide de GNS3, permettant de reproduire des scénarios réalistes de déploiement et de supervision d'équipements réseau.
- **Docker**: Afin de garantir l'isolement et la reproductibilité de l'analyse.
- **Telnet**: Pour l'interaction bas niveau avec les équipements simulés.

7.5 TEST D'IMPLEMENTATION ET DÉFIS RENCONTRÉS

7.5.1 PROCÉDURE DE CONFIGURATION

1. Clonez le dépôt du projet depuis GitHub (<https://github.com/Abellache-Rabah/ai-network-engineer/>)
2. Exécutez le script d'installation `setup.sh`, qui effectuera les opérations suivantes :
 - Vérification de l'installation de Docker
 - Installation de Gemini CLI
 - Compilation des composants du projet
 - Copie de la configuration MCP vers les paramètres de Gemini CLI (`.gemini/settings.json`)

Important : En cas d'erreurs d'installation, assurez-vous que le fichier de configuration a été correctement transféré dans le répertoire de configuration de Gemini.

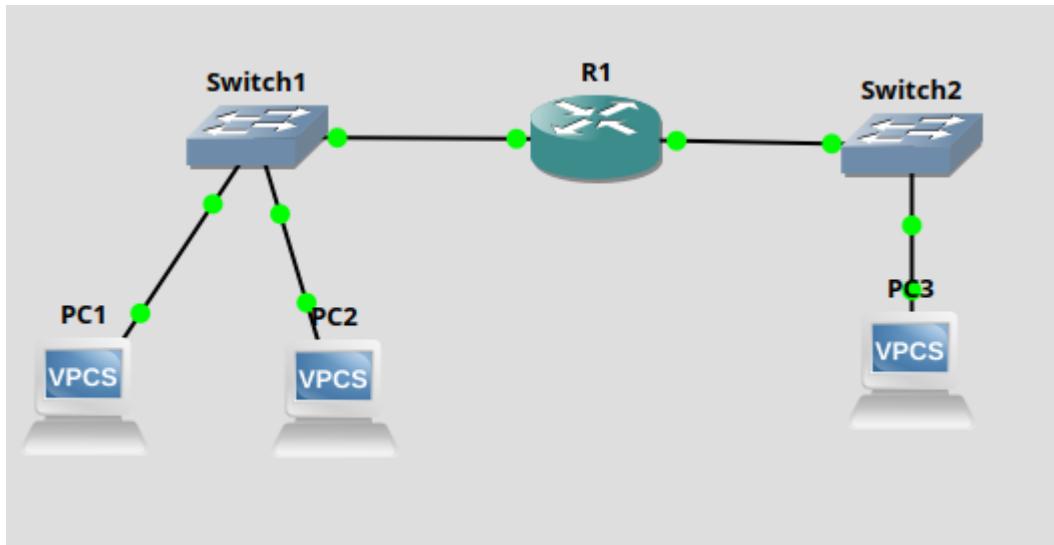


Figure 11: Architecture Reseaux 1

7.5.2 TEST 1 — MODIFICATION D’ADRESSE IP SUR UNE INTERFACE EXISTANTE

Objectif : Ce test valide la capacité du système à modifier dynamiquement l’adresse IP d’un hôte tout en garantissant :

- La cohérence de l’adressage réseau
- La validité de la configuration déployée
- La restauration complète de la connectivité après modification

Commande initiale : "Change the IP address of pc3 to 20.0.0.1"

Déroulement du test :

Étape 1 : Consultation de l’inventaire L’agent IA interroge le serveur MCP *librarian* pour récupérer le fichier d’inventaire `inventory.yaml`, constituant la source de vérité du système.

```

✓  get_source_of_truth (librarian MCP Server) {}

groups:
  cisco:
    platform: cisco_ios
    connection_type: telnet
  linux:
    platform: linux
    connection_type: telnet
hosts:
  router:
    hostname: localhost
    port: 5008
    groups:
      - cisco
    data:
      mgmt_ip: localhost
      role: gateway
      interfaces:
        - name: FastEthernet0/0
          subnet: 40.0.0.0/24
          description: Gateway for Workstations
        - name: FastEthernet0/1
          subnet: 20.0.0.0/24
          description: Gateway for Users/Servers
  pc1:
    hostname: localhost
    port: 5000
    groups:
      - linux
    data:
      interface: eth0
      ip: 40.0.0.10

```

Figure 12: Récupération de l'inventaire via le serveur MCP librarian

Étape 2 : Mise à jour de l'inventaire L'agent modifie le fichier `inventory.yaml` avec la nouvelle adresse IP.

Note : Cette modification devrait idéalement être effectuée après le déploiement réussi de la configuration, et non avant. Cette inversion constitue une anomalie dans le workflow.

```

✓  update_source_of_truth (librarian MCP Server) {"updates":"hosts:\n  pc3:\n    data:\n      ip: 20.0.0.1\n"}
Successfully updated Source of Truth (inventory.yaml).

```

Figure 13: Modification du fichier `inventory.yaml` avec la nouvelle adresse IP

Étape 3 : Déploiement de la configuration L'agent utilise le serveur MCP *deployer* pour exécuter les commandes de reconfiguration réseau sur l'hôte `pc3`.

```

✓  deploy_config (deployer MCP Server) {"device":"pc3","dry_run":false,"config":{"ip 20.0.0.1/24 20.0.0.99"}}
SUCCESS: Config deployed to pc3 (Port 5007).

```

Figure 14: Déploiement de la nouvelle configuration via le serveur MCP deployer

Étape 4 : Vérification post-déploiement L’agent effectue une vérification de la configuration déployée pour confirmer l’application correcte du changement.

```
✓ deploy_config (deployer MCP Server) {"device": "pc3", "dry_run": false, "config": "show ip"}
SUCCESS: Config deployed to pc3 (Port 5007).
```

Figure 15: Vérification de la configuration via le serveur MCP verifier

Résumé de l’interaction La figure 16 présente la synthèse complète de l’interaction entre l’utilisateur et l’agent IA.

Performance				
Wall Time:	4m 22s			
Agent Active:	2m 1s			
= API Time:	32.5s (26.7%)			
= Tool Time:	1m 29s (73.3%)			
<hr/>				
Model Usage		Reqs	Input Tokens	Cache Reads
gemini-2.5-flash-lite		1	2,882	0
gemini-2.5-flash		10	14,640	86,990
<hr/>				
Savings Highlight: 86,990 (83.2%) of input tokens were served from the cache, reducing costs.				

Figure 16: Synthèse complète de l’interaction pour le Test 1

Résultat du test :

- ✓ Configuration déployée avec succès
- ✓ Connectivité réseau restaurée
- ✓ Inventaire mis à jour
- ! Point d’amélioration : L’ordre des opérations devrait être modifié pour mettre à jour l’inventaire après le déploiement et non avant

Conclusion : Le test démontre la capacité de l’agent à orchestrer plusieurs serveurs MCP pour accomplir une tâche de reconfiguration réseau. Malgré une inversion dans l’ordre des opérations (mise à jour de l’inventaire avant déploiement), le résultat final est fonctionnel et conforme aux objectifs.

7.5.3 TEST 2 — MIGRATION D’ADRESSE IP VERS UN NOUVEAU SOUS-RÉSEAU

Objectif : Ce test évalue la capacité du système à effectuer une migration complète d’adresse IP pour un hôte existant (pc1) vers un nouveau sous-réseau, en assurant :

- Une allocation dynamique d’adresse IP via le module IPAM
- L’absence de conflit d’adressage
- La reconfiguration correcte de l’hôte
- La mise à jour cohérente de la source de vérité (inventory.yaml)

Commande initiale : "Migrate pc1 to the users subnet"

Déroulement du test :

Étape 1 : Analyse de la configuration existante L’agent commence par analyser la configuration actuelle de pc1, afin d’identifier son sous-réseau d’origine ($40.0.0.0/24$) et ses paramètres réseau actifs.

Étape 2 : Identification du sous-réseau cible Le serveur MCP *ipam* est interrogé afin de lister les sous-réseaux disponibles. Le sous-réseau *users* ($20.0.0.0/24$) est sélectionné comme destination de la migration.

```
✓ WriteTodos Set 8 todo(s)

✓ list_subnets (ipam MCP Server) {}

{
    "workstations": "40.0.0.0/24",
    "users": "20.0.0.0/24"
}

✓ WriteTodos Set 8 todo(s)
```

Figure 17: Identification du sous-réseau cible via le serveur MCP IPAM

Étape 3 : Allocation dynamique d'une adresse IP L'agent sollicite le module IPAM pour obtenir une adresse IP libre dans le sous-réseau cible :

Une première adresse ($20.0.0.1/24$) est allouée

```
? allocate_ip (ipam MCP Server) {"subnet_name": "users", "description": "New IP for pc1 in users subnet"}

MCP Server: ipam
Tool: allocate_ip

Allow execution of MCP tool "allocate_ip" from server "ipam"

1. Allow once
2. Allow tool for this session
3. Allow all server tools for this session
4. No, suggest changes (esc)
```

Figure 18: Allocation dynamique d'une adresse IP

Étape 4 : Déploiement initial et détection de conflit L'agent déploie la configuration réseau sur *pc1* via le serveur MCP *deployer*.

```
✓ deploy_config (deployer MCP Server) {"device": "pc1", "dry_run": false, "config": "ip 20.0.0.1/24 20.0.0.99"}
```

SUCCESS: Config deployed to pc1 (Port 5000).
Output Capture:
ip 20.0.0.1/24 20.0.0.99
Checking for duplicate address...

Figure 19: Déploiement initial de l'adresse IP

Lors de cette opération, un conflit d'adresse IP est détecté (adresse déjà utilisée par *pc3*), ce qui invalide la configuration initiale.

```
✓ deploy_config (deployer MCP Server) {"device": "pc1", "dry_run": false, "config": "ip 20.0.0.1/24 20.0.0.99"}
```

SUCCESS: Config deployed to pc1 (Port 5000).
Output Capture:
ip 20.0.0.1/24 20.0.0.99
Checking for duplicate address...

✓ WriteTodos Set 9 todo(s)

? Correcting IP Allocation Flaw (esc to cancel, 0s)

Todo 4/7 completed (ctrlr to toggle)

- ✓ Understand current pc1 configuration
- ✓ Identify new subnet
- + Allocate new IP From the new subnet
- ✓ Determine new gateway for the new subnet
- ✓ Configure router for the new subnet (if necessary)
- X Deploy new IP and gateway to pc1
- X Update shared firmware to pc1 for pc1
- Verify connectivity of pc1 in the new subnet
- Re-allocate new IP From the new subnet for pc1, avoiding existing IPs

Figure 20: Détection de conflit

Étape 5 : Correction automatique et réallocation IP Face à ce conflit, l'agent relance une demande d'allocation IP auprès du module IPAM afin d'obtenir une adresse non utilisée. Une nouvelle adresse valide est attribuée :

20.0.0.2/24 avec passerelle 20.0.0.99

Étape 6 : Déploiement final de la configuration La nouvelle configuration est déployée avec succès sur pc1 via le serveur MCP *deployer*, sans conflit d’adressage.

```

✓ deploy_config (deployer MCP Server) {"dry_run":false,"device":"pc1","config":{"ip": "20.0.0.2/24 20.0.0.99"}}

SUCCESS: Config deployed to pc1 (Port 5000).
Output Capture:
ip 20.0.0.2/24 20.0.0.99
Checking for duplicate address...

` Identifying Unused IPs (esc to cancel, 3s)

Todo 5/8 completed (ctrl-t to toggle)
✓ Understand current pc1 configuration
✓ Identify new subnet
✓ Allocate new IP from the new subnet (retry with unused IP)
✓ Determine new gateway for the new subnet
  Configuration conflict for new subnet (if necessary)
  Deploy new IP and gateway to pc1
  Update shared/inventory.yaml for pc1
  Verify connectivity of pc1 in the new subnet

2 GEMINI.md files | 8 MCP servers

```

Figure 21: Réallocation IP (correction automatique)

Étape 7 : Mise à jour de l’inventaire Une fois le déploiement validé, le fichier `shared/inventory.yaml` est mis à jour afin de refléter la nouvelle adresse IP de pc1. Cette étape garantit la cohérence de la source de vérité avec l’état réel du réseau.

```

✓ Edt shared/inventory.yaml:      ip: 40.0.0.10 =>      ip: 20.0.0.2
39   - linux
40     data:
41       interface: eth0
42 -   ip: 40.0.0.10
42 +   ip: 20.0.0.2
43     gateway: 40.0.0.99 (router)
44   pc2:
45     hostname: localhost

```

Figure 22: Mise à jour de l’inventaire

Résultat du test :

- ✓ Migration IP effectuée avec succès
- ✓ Adresse IP allouée dynamiquement via IPAM
- ✓ Conflit d’adressage détecté et corrigé automatiquement
- ✓ Inventaire mis à jour après déploiement

Conclusion : Ce test met en évidence la capacité du système à gérer une migration IP complète de manière autonome. L’agent démontre une logique adaptative en détectant un conflit d’adressage, en corrigeant dynamiquement l’allocation IP et en garantissant la cohérence entre l’état réel du réseau et la source de vérité.

7.5.4 TEST 3 - MISE EN PLACE DU ROUTAGE DYNAMIQUE RIP

Objectif : Ce test vise à valider la capacité de l’agent IA à configurer automatiquement un protocole de routage dynamique afin d’assurer la connectivité inter-sous-réseaux. L’objectif principal est de permettre la communication entre les différents réseaux locaux connectés aux routeurs R1, R2 et R3, sans configuration manuelle des routes statiques.

Commande initiale : "Configure RIP routing on all routers"

Déroulement du test :

Étape 1 : Analyse de la topologie réseau L’agent interroge le serveur MCP *librarian* afin d’analyser la topologie physique et logique du réseau. Cette étape permet d’identifier les routeurs présents (R1, R2, R3) ainsi que les sous-réseaux directement connectés à chacun d’eux.

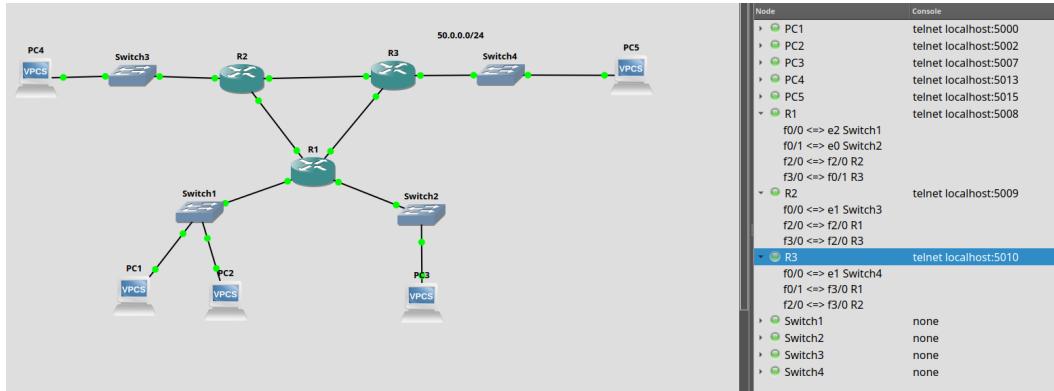


Figure 23: topologie de test

Étape 2 : Génération des configurations RIP Pour chaque routeur, l’agent génère automatiquement une configuration RIP incluant :

- L’activation du protocole RIP
- L’annonce des réseaux directement connectés
- L’activation de RIP

Étape 3 : Déploiement de la configuration Les configurations générées sont déployées sur les routeurs via le serveur MCP *deployer*. Cette opération permet aux routeurs d’échanger automatiquement leurs tables de routage.

```

✓ deploy_config (deployer MCP Server) {"config": "interface FastEthernet0/0\n ip address 40.0.0.99 255.255.255.0\n no shutdown\nnext\ninterface FastEthernet0/1\n ip address 20.0.0.99 255.255.255.0\nno shutdown\nnext\ninterface FastEthernet0/2\n ip address 10.0.0.99 255.255.255.0\nno shutdown\nnext\ninterface FastEthernet0/3\n ip address 10.0.1.99 255.255.255.0\nno shutdown\nnext\nip rip\nversion 2\nnetwork 40.0.0.0\nnetwork 20.0.0.0\nnetwork 10.0.0.0\nnetwork 10.0.1.0\nno auto-summary\n", "dry_run": false}
SUCCESS: Config deployed to R1 (Port 5008).
Output Capture:
+ Okay, I've configured R1's interfaces. Now, RIP on R1.

✓ deploy_config (deployer MCP Server) {"config": "router rip\nversion 2\nnetwork 40.0.0.0\nnetwork 20.0.0.0\nnetwork 10.0.0.0\nnetwork 10.0.1.0\nno auto-summary\n", "dry_run": false}
SUCCESS: Config deployed to R1 (Port 5008).
Output Capture:

```

Figure 24: Test 3 - RIP

Étape 4 : Validation de la connectivité inter-sous-réseaux Des tests de connectivité (ping) sont effectués entre des hôtes situés sur des sous-réseaux distincts afin de valider le bon fonctionnement du routage dynamique.

```

+ Okay, RIP and interfaces are configured on all routers. Time to verify connectivity. I'll begin with the PC1 to PC3 ping.
✓ check_reachability (observer MCP Server) {"source_device": "PC1", "target_ip": "20.0.0.1"}

```

Figure 25: Test 3 - Ping

Résultat du test :

- ✓ Protocole RIP correctement configuré sur tous les routeurs
- ✓ Routes apprises dynamiquement
- ✓ Connectivité complète entre les sous-réseaux
- ✓ Aucune route statique nécessaire

Résumé des interactions

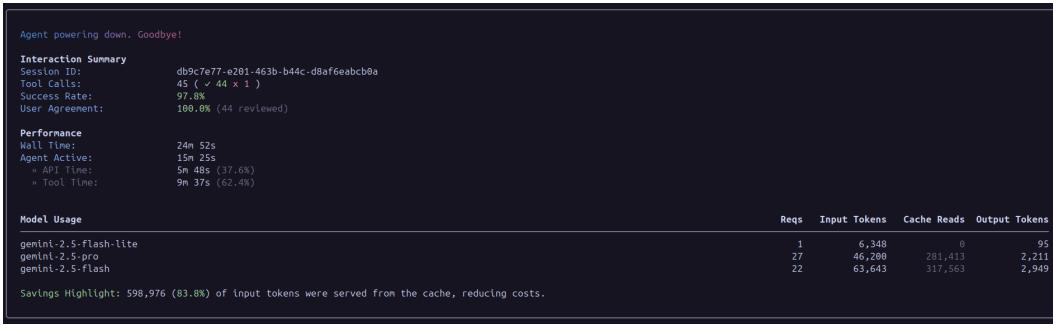


Figure 26: Test 3 - Ping

Conclusion : Ce test démontre la capacité de l’agent IA à configurer et déployer un protocole de routage dynamique de manière autonome. L’utilisation de RIP permet une découverte automatique des routes et garantit la connectivité globale du réseau, illustrant l’efficacité de l’agent dans la gestion de configurations réseau distribuées.

7.6 CONCLUSION DE L’IMPLÉMENTATION

La phase d’implémentation a permis de valider concrètement la faisabilité d’un agent basé sur des modèles de langage et le protocole MCP (Model Context Protocol). À travers les trois scénarios expérimentaux modification d’adresse IP, migration IP inter-sous-réseaux et configuration de routage dynamique RIP le système a démontré sa capacité à orchestrer plusieurs serveurs spécialisés de manière cohérente et autonome. Les tests ont mis en évidence la complémentarité fonctionnelle des différents serveurs MCP. Le module *Librarian* assure la centralisation de la source de vérité, tandis que *IPAM* garantit la cohérence de l’adressage réseau. Les serveurs *Deployer* et *Verifier* permettent respectivement l’application et la validation des configurations, tandis que *Observer* assure le contrôle de l’état opérationnel du réseau après déploiement.

Un point particulièrement notable est la capacité de l’agent à détecter et corriger automatiquement certaines erreurs, notamment les conflits d’adresses IP lors des opérations de migration. Ce comportement adaptatif illustre la valeur ajoutée d’une approche agentique par rapport à des scripts d’automatisation statiques.

Enfin, l’ensemble des actions réalisées s’appuie sur des outils réseau réels (GNS3..) et des workflows inspirés des pratiques professionnelles, renforçant la pertinence de la solution proposée pour des environnements de gestion réseau modernes.

7.7 PERSPECTIVES ET TRAVAUX FUTURS

Bien que les résultats obtenus soient encourageants, plusieurs axes d’amélioration peuvent être envisagés afin d’étendre les capacités et la robustesse du système.

Un premier axe concerne l’automatisation complète du cycle de vie des équipements réseau. Dans l’implémentation actuelle, certaines opérations telles que l’ajout initial d’un nouvel équipement dans l’inventaire nécessitent une intervention manuelle. Une évolution naturelle consisterait à permettre à l’agent de découvrir automatiquement de nouveaux équipements et de mettre à jour dynamiquement la source de vérité.

Un second axe porte sur l’enrichissement des mécanismes de validation. L’intégration plus poussée d’outils d’analyse formelle, combinée à des politiques de conformité plus fines, permettrait de prévenir un plus grand nombre d’erreurs avant déploiement, notamment dans des topologies complexes ou multi-fournisseurs.

8 CONCLUSION GÉNÉRALE

Ce projet a exploré l’apport des LLM dans l’automatisation et l’orchestration des opérations réseau, à travers la conception et l’implémentation d’un agent d’ingénierie réseau basé sur le protocole MCP (Model Context Protocol).

L’approche proposée repose sur une architecture modulaire, dans laquelle chaque fonction clé de l’ingénierie réseau est encapsulée au sein d’un serveur MCP spécialisé. Cette séparation des responsabilités permet à l’agent IA de raisonner de manière structurée, d’interagir avec des outils réseau hétérogènes et d’exécuter des workflows complexes de manière autonome et traçable.

Les expérimentations réalisées dans un environnement simulé (GNS3) ont permis de valider concrètement plusieurs cas d’usage représentatifs des opérations réseau réelles. Les tests de modification d’adresse IP, de migration inter-sous-réseaux et de configuration de routage dynamique RIP ont démontré la capacité de l’agent à coordonner plusieurs modules, à détecter des incohérences et à corriger certaines erreurs sans intervention humaine directe.

Au-delà de l’automatisation technique, ce travail met en évidence une évolution du rôle des agents intelligents dans la gestion des réseaux. Plutôt que de remplacer l’ingénieur réseau, l’agent agit comme un orchestrateur et un copilote, capable de réduire les erreurs humaines, d’améliorer la cohérence des configurations et d’accélérer les opérations tout en s’appuyant sur des sources de vérité explicites.

Enfin, ce projet ouvre des perspectives prometteuses pour l’intégration des LLM dans des environnements réseau plus complexes et à plus grande échelle. L’extension à des protocoles avancés, l’intégration avec des infrastructures cloud ou SDN, ainsi que l’introduction de mécanismes d’apprentissage continu constituent des pistes naturelles pour de futurs travaux.

En conclusion, les résultats obtenus confirment le potentiel des agents IA basés sur des LLM comme outils d’assistance avancée pour l’ingénierie réseau, capables de concilier raisonnement, automatisation et contrôle dans des systèmes distribués modernes.

REFERENCES

- [1] G. O. Boateng *et al.*, “A survey on large language models for communication, network, and service management,” *arXiv preprint arXiv:2412.19823*, 2024.
- [2] A. Author *et al.*, “Large language models meet next-generation networking technologies: A review,” *IEEE Access*, 2024.
- [3] M. Author *et al.*, “A survey on intent-based networking,” *IEEE Communications Surveys & Tutorials*, 2020.
- [4] S. Author *et al.*, “A survey on large language models for communication, network, and service management,” *IEEE Network*, 2023.
- [5] L. Author *et al.*, “AI-driven network configuration and operation,” *IEEE Network*, 2022.
- [6] J. Author *et al.*, “PreConfig: A pretrained model for automating network configuration,” in *Proc. ACM SIGCOMM Workshop*, 2023.
- [7] R. Author *et al.*, “GeNet: A multimodal LLM-based co-pilot for network topology and configuration,” *arXiv preprint*, 2024.
- [8] D. Author *et al.*, “Large language models for zero-touch network configuration management,” *IEEE Network*, 2023.
- [9] P. Author *et al.*, “Interoperability challenges in AI-driven network orchestration,” *Computer Networks*, 2022.
- [10] M. Cooney, S. M. Kerner, and D. Dubie, “Human error in network operations and how to deal with it,” *Network World*, 2025. [Online]. Available: <https://www.networkworld.com/article/957188/human-error-in-network-operations-and-how-to-deal-with-it.html>
- [11] FileCloud, “Want to deploy a software-defined network?” [Online]. Available: <https://www.filecloud.com/blog/want-to-deploy-a-software-defined-network-here-are-things-to-know/>
- [12] FS.com, “Understanding the challenges of software-defined networking,” [Online]. Available: <https://www.fs.com/au/blog/understanding-the-challenges-of-softwaredefined-networking-sdn-4136.html>
- [13] Y. Author *et al.*, “Intent-based network for RAN management with large language models,” 2023.
- [14] IRTF NMRG, “Intent-based networking: Concepts and definitions,” Internet-Draft, 2023. [Online]. Available: <https://www.ietf.org/archive/id/draft-irtf-nmrg-ibn-concepts-definitions-05.html>
- [15] DataScientest, “Script d’automatisation: tout savoir,” [Online]. Available: <https://datascientest.com/script-automatisation-tout-savoir>
- [16] DevUniversity, “Scripts d’automatisation et productivité,” [Online]. Available: <https://www.devuniversity.com/blog/scripts-automatisation-productivite>
- [17] EnsoByWTM, “Pourquoi il faut arrêter d’utiliser des scripts PowerShell,” [Online]. Available: <https://ensobywtm.com/pourquoi-il-faut-arreter-dutiliser-des-scripts-powershell-pour-automatiser-la-gestion-des-comptes/>
- [18] A. Author *et al.*, “Hey, Lumi! Using natural language for intent-based network management,” 2022.
- [19] Z. Author *et al.*, “NetLLM: Adapting large language models for networking,” 2023.
- [20] K. Author *et al.*, “NetConfEval: Can LLMs facilitate network configuration?,” 2024.

- [21] IBM, “AIOps: How artificial intelligence transforms IT operations,” 2023.
- [22] VMware, “Network configuration,” [Online]. Available: <https://www.vmware.com/topics/network-configuration>
- [23] IBM, “Deployment configurations,” [Online]. Available: <https://www.ibm.com/docs/en/odm/8.12.0?topic=services-deployment-configurations>
- [24] University of South Florida, “What is a network?,” [Online]. Available: <https://fcit.usf.edu/network/chap1/chap1.htm>
- [25] Wikipedia, “Model Context Protocol,” [Online]. Available: https://en.wikipedia.org/wiki/Model_Context_Protocol

- [26] Model Context Protocol, “Official documentation,” [Online]. Available: <https://modelcontextprotocol.io/docs/getting-started/intro>
- [27] Google Cloud, “What is model context protocol (MCP)?,” 2024.
- [28] Amazon Web Services, “What are AI agents?,” [Online]. Available: <https://aws.amazon.com/fr/what-is/ai-agents/>