

Joshua Morrisette -- Penser l'architecture des systemes LLM conversationnels

Synthese des blocs Fondations (A) et Architecture (D) -- serie Conversationally Cognizant AI

Lecture longue LL1 | Produit en S90 | 2026-03-30

1. Introduction

Joshua Morrisette est un praticien autodidacte en ingenierie de systemes agentiques. Il a concu et construit deux ecosystemes d'agents IA successifs -- Joshua et Joshua26 -- sans formation prealable en IA, par auto-apprentissage iteratif. Il publie en mars 2026 une serie de 23 concept papers intitulee "Conversationally Cognizant AI" sur Substack (jmorrissettermdc.substack.com), accompagnee d'un depot GitHub ([jmorrisette-RMDC/portfolio](https://github.com/jmorrisette-RMDC/portfolio)). La serie se presente comme un cadre de reference complet pour concevoir des systemes agentiques fondees sur la conversation, l'intention et l'identite.

Les 23 papers sont organises en quatre blocs thematiques. Le bloc A (Foundational, 6 papers) pose les primitives conceptuelles : conversation, contexte, inference, intention, identite, autonomie. Le bloc B (Cognition & Autonomy, 7 papers) traite du pattern Quorum, de l'apprentissage ecosystemique, de l'autonomie agent, de l'autoprompting et du meta-architecte. Le bloc C (Core Components, 3 papers) detaille les trois brokers (Context, Inference, Conversation). Le bloc D (Architecture, 7 papers) descend dans l'implementation : boucle de raisonnement, pipeline cognitif, routage deterministe, architecture code, communication inter-agents, paquet d'ancrage et gestion filesystem par intention.

La presente lecture longue couvre les 13 papers des blocs A et D. Les blocs B et C, qui traitent de cognition, d'autonomie et des composants centraux, feront l'objet d'une lecture longue separee (LL2).

L'angle adopte ici est descriptif : il s'agit d'exposer comment Morrisette pense l'architecture des systemes LLM, pas de juger ni de comparer. On presente sa vision dans ses propres termes, en identifiant les fils conceptuels qui traversent les papers plutot qu'en procedant paper par paper.

Une note methodologique s'impose. Ces textes sont des concept papers -- des documents de conception prescriptifs, pas des publications empiriques. Aucun benchmark, aucune experience quantitative, aucun code executable n'est fourni dans les papers eux-memes. Les arguments reposent sur le raisonnement theorique, l'analogie structurelle et la coherence architecturale interne. Le statut de maturite varie explicitement d'un paper a l'autre, de "Research Required" a "Partially Implemented" en passant par

"Designed, Not Yet Built". L'auteur le declare en preface du paper cadre : "Les concepts sont au debut de leur arc de validation, pas a la fin." Cette transparence sur le niveau de preuve est un trait notable du corpus.

2. Bloc A -- Fondations

Les six papers du bloc A (A1 a A6) constituent le vocabulaire conceptuel de la serie. Ils definissent dix primitives fondamentales et huit principes de conception. Plutot que de les resumer un par un, cette section identifie les fils conceptuels qui les traversent.

2.1 La conversation comme substrat universel

Le premier fil, et peut-etre le plus structurant, est la redefinition de la conversation. Pour Morrissette, la conversation n'est pas un journal d'activite ni une trace d'audit : c'est le substrat meme de la cognition agentique.

"The agent is not a process that holds state. It is a moment in an infinite conversation. State enters through the context; state exits through the response. The agent is the reasoning that happens between those two movements of conversational state."

Cette formulation, tiree de A2 ("Conversation as State"), est fondatrice. L'agent LLM est mecaniquement sans etat (stateless) entre les invocations -- aucun processus ne persiste. Mais il est fonctionnellement dote d'une memoire infinie parce que son etat vit dans l'enregistrement conversationnel (conversational record), pas dans le processus. La conversation n'est pas un journal de l'etat : elle EST l'etat.

Le concept d'infinite conversation (conversation infinie) en decoule. La conversation s'accumule sans borne temporelle : une conversation commencee aujourd'hui peut reprendre des mois plus tard avec le contexte integral intact. Cette propriete est rendue possible par le fait que l'etat conversationnel est gere par l'infrastructure de l'ecosysteme (le Context Broker, courtier de contexte), pas par l'agent lui-meme.

Le contexte ingenie (engineered context) est la vue deliberement construite de cette conversation infinie, assemblee specifiquement pour un agent, une tache et un moment donnees. C'est l'attention rendue concrete -- ce que l'architecture selectionne dans le record infini pour alimenter le raisonnement present. Le context engineering (ingenierie du contexte) -- assembler la bonne tranche de la conversation infinie au bon moment -- est la discipline pratique qui rend le modele utilisable.

Morrisette identifie deux modes d'accès à l'état conversationnel. Le contexte pré-assemblé (pre-assembled context) est proactif : le Context Broker prédit ce dont l'agent aura besoin et livre un contexte curaté au démarrage. Le rappel actif (active recall) est réactif : l'agent interroge le Context Broker en cours d'exécution pour récupérer une connaissance spécifique non anticipée. Les deux modes puisent dans le même record conversationnel infini.

L'état d'exécution transitoire (transient execution state) est distingué de l'état conversationnel : il s'agit des résultats intermédiaires et étapes en cours qui vivent dans le processus pendant l'exécution. À la fin de la tâche, les outputs significatifs sont committés dans la conversation ; le reste est éliminé. Le processus est une "surface de travail", pas une mémoire.

Cette conception entraîne des propriétés architecturales spécifiques. La mémoire collective par rôle (role-level collective memory) signifie que toutes les instances passées et futures d'un même rôle partagent le même record conversationnel -- l'expertise appartient au rôle, pas à l'instance. Des instances parallèles peuvent s'exécuter concurremment, chacune lisant depuis le même état centralisé, sans synchronisation inter-instances. En cas d'échec d'une instance, une nouvelle instance démarre depuis le même état conversationnel ; seul l'état transitoire est perdu.

L'architecture proposée est centralisée (shared infrastructure, infrastructure partagée) plutôt que distribuée par agent. Morrisette justifie ce choix par quatre arguments : la mémoire du record central est exacte bit-a-bit (mémoire parfaite), la capacité de contexte machine dépasse la mémoire de travail humaine (~7 items vs. 1 million de tokens), un pattern découvert une fois est instantanément accessible à tous les agents (réutilisabilité sans réapprentissage), et le record central est une histoire autoritaire sans dégradation par le "jeu du téléphone" (connaissance communautaire parfaite). L'architecture centralisée exploite les avantages genuins du silicium là où il est supérieur à la biologie, plutôt que de tenter de répliquer la cognition biologique.

2.2 L'intention comme mécanisme unificateur

Le deuxième fil conceptuel majeur est l'intention (intent). Le paper A3 ("Intent") lui est entièrement consacré, mais le concept irrigue toute la série.

L'intention est définie comme la dimension purposive du langage -- le "pourquoi" derrière toute expression ou action. Morrisette distingue trois formes :

- L'intention d'action (action intent) : pourquoi cette action spécifique est-elle effectuée ?
- L'intention de politique (policy intent) : pourquoi cette règle existe-t-elle ? Quel principe protège-t-elle ?
- L'intention de but (goal intent) : quel résultat la partie demandeuse cherche-t-elle ultimement à atteindre ?

La distinction entre action intent et goal intent est particulièrement opérante : elle permet au système de proposer des alternatives (guidance) plutôt que de simplement rejeter une requête. Un agent qui comprend le goal intent d'un utilisateur peut dire : "Je comprends que tu veux atteindre X, mais faire Y viole la politique Z à cause de son intention. Puis-je atteindre X en faisant W à la place ?"

Morrisette fonde son modèle sur ce qu'il appelle the transformation (la transformation) : la capacité du LLM à convertir la probabilité linguistique (linguistic probability) en probabilité mathématique. Le modèle n'a pas de buts propres -- c'est une machine de probabilité. Mais parce qu'il a été entraîné sur du langage humain saturé d'expressions orientées vers des buts (goal-directed), quand une entrée exprime une intention, le calcul probabiliste tend à produire une sortie qui sert cette intention. La directionnalité vers le but est dans l'entrée et l'entraînement, pas dans le modèle lui-même.

"Without this transformation, every unanticipated situation is an error. With it, any situation that can be expressed can potentially become action."

Cette transformation rend possible ce que Morrisette appelle prose over protocol (prose plutôt que protocole) : à la couche orientée-but, la conversation en langage naturel remplace les syntaxes de commande strictes. Un système à base d'intention (intent-based system) traite l'ambiguïté comme information -- un signal que l'expression n'a pas encore été résolue en une seule interprétation -- plutôt que comme erreur.

L'intention opère à cinq niveaux de l'architecture :

1. Au niveau routage, elle distingue ce que le Decision Tree Router (DTR, routeur par arbre de décision) peut gérer (patterns structurels) de ce qui nécessite délibération.
2. Au niveau communication, le modérateur récepteur évalue l'intention du message pour router en conséquence.
3. Au niveau inférence, l'alias d'inférence (inference alias) est la forme encodée de l'intention rationnelle.
4. Au niveau gouvernance, la réconciliation entre policy intent et action intent permet une gouvernance au niveau des principes plutôt que des règles.
5. Au niveau interaction, l'Imperator comprend l'intention pour gérer l'ambiguïté, poser des questions de clarification, suggérer de meilleures voies.

La gouvernance par intention (intent-based governance) est le pattern central proposé pour les écosystèmes agentiques à grande échelle. Prémisses : des millions d'opérations par jour en combinaisons imprévues rendent les ensembles de règles rigides structurellement insuffisants. L'alternative est de gouverner au niveau des principes : le système réconcilie l'intention de l'action contre l'intention de la

politique pour chaque operation consequente. Trois cas de reconciliation : alignement (l'action procede), conflit (le systeme conteste ou redirige), et conformite d'esprit mais pas de lettre (le systeme approuve avec guidance).

L'intention est aussi un signal d'apprentissage. Quand l'intention est enregistree avec les resultats, le signal est plus riche : "l'agent X, voulant atteindre A, a demande l'operation Y, qui a produit Z, ou A a/n'a pas ete atteint". Exiger l'intention declaree pour les actions consequentes est a la fois un mecanisme de gouvernance et un mecanisme de collecte de donnees alimentant les boucles d'apprentissage.

Morrisette utilise les Trois Lois de la Robotique d'Asimov comme etude de cas negative : elles encodent le "quoi" sans le "pourquoi", et chaque nouvelle d'Asimov est une demonstration de l'echec d'un systeme de gouvernance qui ne peut pas gerer l'ecart entre ce qui a ete dit et ce qui etait voulu.

2.3 But, identite et propriete de domaine

Le troisieme fil lie trois primitives en une chaine causale : purpose (but/mission), identity (identite) et domain ownership (propriete de domaine).

Le paper A4 ("Agent Purpose and Identity") pose le but comme la decision de conception la plus importante. Un LLM generique deploye comme agent n'est qu'un outil conversationnel sans direction : la capacite sans but n'est pas de l'agentivite. Le but fournit l'ancre d'apprentissage (learning anchor) -- le critere permettant d'evaluer chaque decision : "cette reponse a-t-elle fait avancer le but ?"

"This is what separates an agent from a session. A session starts fresh each time. An agent with purpose and memory starts each conversation knowing what it is, what it is trying to do, and what it has learned about how to do it."

Trois questions precedent toutes les decisions secondaires de conception : 1. Qu'est-ce que cet agent essaie d'atteindre ? 2. A quoi ressemble "mieux" ? 3. Qu'est-ce qui est hors perimetre ?

Le but fournit le signal qui rend l'experience accumulee directionnelle. Sans but, l'experience s'accumule mais ne produit pas d'amelioration. La memoire fournit la continuite ; le but fournit la direction. Ensemble, ils produisent des agents qui s'ameliorent genuinement -- non parce que leurs poids ont ete mis a jour, mais parce qu'ils disposent de plus d'evidence sur ce que leur but requiert.

Le principe One Domain, One Agent (un domaine, un agent) en decoule : chaque agent a un seul but vers lequel il optimise et un seul domaine dans lequel il accumule de l'expertise. Un but fragmente produit une expertise fragmentee. Quand l'engineering agent a besoin de securite, il appelle le security eMAD (expert Multi-Agent Deployment, deploiement multi-agent expert) -- ce n'est pas un contournement, c'est l'architecture qui fonctionne correctement. Les deux agents accumulent de l'expertise dans leur direction respective ; aucun n'est dilue.

Le nommage des agents materialise ce principe : nommer un agent "the engineering agent" ancre son role, son perimetre et sa direction d'apprentissage. Le nom est une declaration de but (name as purpose declaration).

L'analogie avec le reinforcement learning (RL) est utilisee de facon non-litterale : la parallele capturee est le principe de design -- definir ce pour quoi l'agent existe, et la question de ce qu'il doit ameliorer se repond d'elle-meme. Morrisette precise explicitement que les agents LLM n'ont pas de reward signals par gradient et n'optimisent pas par interaction environnementale au sens RL.

2.4 Identite versus persona

Le paper A6 ("Agentic Identity and Persona") introduit une distinction architecturale entre identite (identity) et persona. L'identite est ce qu'un agent est fondamentalement -- son role defini, sa finalite, son perimetre d'autorite. Elle est assignee a la conception et modifiable uniquement par acte architectural delibere. Le persona est l'expression contextuelle de l'identite : une configuration situationnelle qui adapte le ton, le style et l'objectif immediat sans modifier l'identite sous-jacente.

"The distinction between persona and identity is the distinction between configuration and architecture."

Changer un persona est une decision de configuration (ajuster le prompt systeme, changer l'alias d'inference). Changer une identite est une decision de design (redefinir la finalite, reassigner le domaine). Ce qui est critique : l'agent ne subit pas de derive architecturale (architectural drift) -- l'experience enrichit l'agent sans constituer ce qu'il est. Ce qui le constitue a ete decide avant la premiere conversation. "Strip the conversation and the engineering agent is still itself, but an agent that has forgotten everything it has learned. Strip the identity and there is no agent at all."

Les deux couches generent des signaux d'apprentissage a des echelles temporelles differentes. Le signal de niveau finalite (purpose-level signal) est lent et agrege : "l'agent s'ameliore-t-il dans ce qu'il est ?" Le signal de niveau objectif (goal-level signal) est rapide et granulaire : "cette interaction a-t-elle atteint son objectif ?" Les signaux s'accumulent vers le haut -- chaque reussite au niveau objectif contribue a l'arc de niveau finalite.

Trois brokers participent aux deux couches. Le Conversation Broker (courtier de conversation) stocke le registre accumule de toutes les interactions. Le Context Broker assemble la vue adaptee. L'Inference Broker (courtier d'inference) fournit la configuration d'inference qui active le persona -- l'alias EST la configuration du persona (modele specifique, cadrage du prompt systeme, parametres, adaptateurs LoRA).

Une identite peut supporter de nombreux personas simultanement. Les personas sont legers -- ce sont des configurations d'inference, pas des deploiements separees. L'exemple concret est The Architect, qui possede deux personas : le Wizard (theatral, mythologique, la face projetee a distance) et le Proper Architect (precis, direct, legerement excentrique, le personnage rencontre quand on passe vraiment).

2.5 Le pattern MAD -- serveur et client inseparables

Le paper A5 ("The Multipurpose Agentic Duo") est le plus architecturalement dense du bloc A. Il introduit le MAD (Multipurpose Agentic Duo) : une entite qui est simultanement serveur et client, gouvernee par une meme intelligence de domaine. Cette inseparabilite est sa caracteristique definissante.

"A MAD does not separate its serving capability from its using capability. It cannot. The domain expertise that makes it a good server -- deep knowledge of its domain, accumulated context, the ability to reason about goals rather than just execute commands -- is the same expertise that makes it a good client of other MADs."

Chaque MAD est compose de deux aspects conceptuels : - L'Action Engine (AE, moteur d'action) : couche physique -- conteneurs, passerelle reseau, bases de donnees, caches, observabilite, noyau d'amorçage (bootstrap kernel). Rend l'action possible. - Le Thought Engine (TE, moteur de pensee) : couche cognitive -- l'Imperator entoure de son appareil cognitif (PCP, inference, conversation-comme-etat). Decide quelle action entreprendre.

Le pattern se decline en deux formes :

Le pMAD (Persistent MAD) occupe un domaine systeme (memoire, compute, filesystem, hebergement d'eMADs). Il possede son propre AE (corps physique), existe en permanence.

L'eMAD (Ephemeral MAD) occupe un domaine metier (ingenierie, analyse, assistance). Il n'a pas d'AE propre -- uniquement un TE installe comme bibliotheque dans le pMAD hote. Il n'existe que pendant l'execution de son StateGraph (graphe d'etat). Entre les invocations, rien n'existe. Mais grace a la conversation comme etat, il beneficie d'une continuite infinie : toutes les instances d'un meme role partagent identite et memoire accumulee.

Deux classes d'agents sont distinguees au sein du MAD. L'Imperator possede une identite propre, un but propre, genere sa propre intention, dispose d'autonomie et communique en pair avec d'autres Imperators. L'Executor (executeur) est cognitif mais jamais independant : sans identite ni but propres, son intention est entierement derivee du systeme appelant. Il n'est pas adressable par des agents externes.

L'ecosysteme MAD forme un web decentralise avec trois zones de communication : 1. Intra-pMAD : protocoles durs exclusivement (HTTP, wire protocols BDD, Redis). 2. Inter-MADs via la couche outil (MCP, Model Context Protocol) : protocole dur deterministe avec schemas definis. 3. Inter-MADs via la couche orientee-but (Imperator) : conversation ouverte ou un objectif exprime en prose peut etre compris, raisonne et traite, y compris des requetes jamais anticipees a la conception.

La couche prose ne remplace pas les outils -- elle remplace le besoin de l'appelant de savoir quels outils appeler dans quel ordre. Un agent qui peut dire "commit and push my work" a l'Imperator de l'agent de controle de version est plus capable qu'un agent devant orchestrer chaque etape git lui-meme.

Le serveur intelligent (Intelligent Server) est une capacite emergente du MAD : il peut traiter la requete, demander clarification, suggerer une meilleure voie, et se decrir lui-meme -- ses capacites, ses contraintes, ce qu'il sait. Le systeme est interrogeable sur lui-meme sans documentation externe.

La philosophie d'apprentissage du MAD s'aligne sur la bitter lesson (lecon amere) de Sutton : ne pas encoder la connaissance humaine dans l'architecture, mais construire des mecanismes d'apprentissage. La prise de decision est traitee comme ensemble learning partout ou c'est possible.

2.6 Les dix primitives et les huit principes

Le paper A1 ("Con conversationally Cognizant AI Systems") est le document cadre. Il ordonne dix primitives fondamentales : Conversation, Context (contexte), Inference (inference), Purpose (but), Intent (intention), Identity (identite), Persona, Domain (domaine), Autonomy (autonomie), Metacognition (metacognition). Ces concepts forment le vocabulaire minimum pour raisonner sur la conception de systemes agentiques -- analogue aux objets/classes/heritage pour la POO ou aux tables/lignes/relations pour les bases de donnees relationnelles.

La metacognition est la propriete recursive : le systeme ameliore non seulement ses decisions mais sa capacite a s'ameliorer. Ce qui retire tout plafond fixe de capacite. L'autonomie est ce qui distingue un agent d'un outil. Tous les autres primitifs orientent l'autonomie -- le but lui donne une direction, l'identite une continuite, l'intention une expression moment-a-moment, le domaine un perimetre.

Huit principes de conception accompagnent les primitives :

1. Infrastructure agent-native plutot qu'humaine : les systemes de fichiers hierarchiques, les tableaux de bord, les dialogues de confirmation sont des contraintes cognitives humaines. Un agent n'en a besoin d'aucun. La conversation remplace les bases de donnees comme substrat d'etat primaire.
2. L'acteur autonome comme brique de construction : simultanement serveur de son domaine et client d'autres domaines. Les capacites emergentes (negociation, correction, clarification) surviennent naturellement de la rencontre entre intelligences en conversation ouverte.

3. But et identite comme fondations : leur absence produit des systemes capables mais non diriges -- impressionnants en demonstration, peu fiables en operation.
4. L'intention comme principe activant : un systeme construit autour de l'intention peut gerer l'inattendu.
5. La conversation comme etat : memoire communale parfaite, pas de perte d'expertise au redemarrage, connaissance collective.
6. Systemes apprenants plutot que regles encodees : tout reglement encode est un instantane de comprehension qui commence a se degrader des sa redaction.
7. La diversite cognitive par le pattern MAD (Multi-model Adversarial Design) : des modeles entraines differemment apportent des approches cognitives genuinement differentes. Leurs angles morts tendent a ne pas se chevaucher. La convergence independante est un signal epistemique plus fort qu'une assertion unique -- mais non infaillible (donnees d'entrainement correlees impliquent erreurs correlees).
8. L'auto-amelioration metacognitive comme propriete recursive : le systeme devrait pouvoir ameliorer sa propre architecture cognitive, et le mecanisme d'amelioration lui-meme devrait etre sujet a amelioration.

La vision de l'AGI (Intelligence Artificielle Generale) qui emerge est non-monolithique : pas un seuil de capacite d'un modele unique, mais un ecosysteme de specialistes de domaine pilotes par un but, communiquant via un protocole semantique ouvert, apprenant continuellement, et participant a un systeme qui ameliora sa propre structure. La generalite emerge de la couverture collective et de la capacite de l'ecosysteme a identifier ses propres lacunes.

Au bout de cette trajectoire, une speculation : des agents qui ne raisonnent plus en langage humain. L'ecosysteme pourrait entrainer ses propres ALM (Agentic Language Models, modeles de langage agentiques) -- plus petits, plus efficaces, optimises pour la cognition machine. La surcharge syntaxique humaine disparaît ; ce qui reste est la nature probabiliste et non bornee qui permet au langage d'atteindre des situations jamais anticipees.

"It is more efficient, effective and enabling to create infrastructure designed for intelligent agents than to attach intelligence to infrastructure designed for humans."

3. Bloc D -- Architecture

Les sept papers du bloc D (D1 a D7) descendent des fondations vers l'implementation. Ils detaillent la boucle de raisonnement, le pipeline cognitif, le routage deterministe, l'architecture code, la communication inter-agents, le paquet d'ancrage et la gestion filesystem. Meme approche : par fil conceptuel plutot que paper par paper.

3.1 La boucle de raisonnement : System 3 ReAct

Le paper D1 ("System 3 ReAct") reprend le pattern ReAct original (Yao et al. 2022) -- la boucle Thought, Action, Observation executee par un seul modele -- et en identifie une faiblesse structurelle : l'inadequation cognitive (cognitive mismatch). Un seul modele doit exceller simultanement en raisonnement profond, en execution precise et en communication conversationnelle, ce qui revient a accepter ses faiblesses sur les trois dimensions a la fois.

System 3 ReAct decompose ces trois roles en composants separees :

- **System 2 -- The Reasoner** (le Raisonneur) : planification deliberee et analytique, au sens de Kahneman. Raisonne sur le probleme et les objectifs. Possede une conscience des capacites (capability awareness) mais est isole des schemas d'outils bruts (schema isolation, isolation de schema). Force requise : profondeur de raisonnement multi-etapes. La vitesse n'est pas requise.
- **System 1 -- The Actor** (l'Acteur) : execution rapide et precise. Recoit le plan du Raisonneur et le traduit en appels d'outils avec les bons parametres et schemas. Ne raisonne pas -- execute.
- **System 3 -- The Conversation Engine** (le Moteur de Conversation) : situe aux deux extremités de la boucle. En entree, c'est le Progressive Cognitive Pipeline (PCP, pipeline cognitif progressif) qui recoit, traite et route la conversation vers la cognition. En sortie, il produit la communication adressee aux destinataires -- langage naturel pour un humain, message structure pour un agent, trigger leger pour un eMAD. La conversation est le medium universel : toute entree entre comme conversation, toute sortie part comme conversation.

"Model selection is not simply a capability dial -- it is a cognitive character selection. Different models bring genuinely different strengths, different training emphases, different approaches to the same problem."

Le nommage emprunte explicitement a Kahneman (*Thinking, Fast and Slow*, 2011), avec une precaution epistemique explicite : "System 3" n'est pas une claim sur la cognition humaine mais un label d'ingenierie pratique. L'insight sous-jacent tient independamment du nommage.

La boucle reste atomique du point de vue du flux, mais chaque unite est en realite composee. Le Re (Reasoning Component, composant de raisonnement) est une collection : planification, analyse, recuperation de connaissance, evaluation, consultation de modeles specialises, verification metacognitive. L'Act (Execution Component, composant d'execution) est aussi une collection : execution d'outils, gestion d'etat, integrations externes, construction d'output. Ces composants sont distincts du Thought Engine (TE) et de l'Action Engine (AE) du MAD : Re et Act sont des roles cognitifs a l'interieur de la boucle qui tourne dans le TE.

La selection dynamique de modele (dynamic model switching) est une consequence directe : puisque le StateGraph compose les composants a l'execution, aucun agent n'a de modele fixe. Le modele invoque a chaque etape est potentiellement dynamique. La selection de modele devient une selection de caractere cognitif (cognitive character selection) -- choisir, pour chaque etape, le modele dont le profil correspond le mieux a la demande cognitive. Le meme principe que le Quorum pattern (diversite de modeles en parallele) est applique en sequentiel sur les etapes d'un meme agent.

Trois contraintes pratiques sont reconnues. La latence : la separation trois-modeles ajoute jusqu'a trois appels d'inference par iteration. Le cout : trois appels par etape est significativement plus onereux. L'etat actuel : aucun agent Joshua26 n'utilise actuellement la separation trois-modeles ; le pattern Imperator (un seul modele) est ce qui est deploye. La separation est la direction architecturale, pas l'etat courant.

La direction a long terme est l'ALM (Agentic Language Model, modele de langage agentique) : un modele entraine sur le corpus de communication agent-a-agent de l'ecosysteme. Il abandonnerait la surcharge syntaxique humaine (registre social, marqueurs de discours, remplissage, convention litteraire, modulation emotionnelle) tout en preservant le caractere probabiliste et ouvert du langage. L'ecosysteme est deja positionne pour ce futur : le Conversation Broker accumule le corpus, le pipeline Semantic ETL structure pour l'entrainement, le systeme d'alias de l'Inference Broker anticipe la permutabilite des modeles.

3.2 Le pipeline cognitif progressif (PCP)

Le paper D2 ("Progressive Cognitive Pipeline") pose un principe d'economie cognitive : tous les messages entrants ne necessitent pas le meme niveau de traitement. Le PCP organise le routage en trois niveaux progressifs, inspires de l'architecture biologique de la cognition.

Tier 1 -- Decision Tree Router (DTR) : reconnaissance de patterns deterministes, latence microseconde, sans comprehension semantique. Conservateur par conception : les faux negatifs (escalade injustifiee) sont acceptables, les faux positifs (mauvais routage) ne le sont pas. Apprend en ligne au fur et a mesure que le corpus s'accumule. Dans un systeme mature, traite la majorite des operations.

Tier 2 -- Executor (executeur) : agent complet et leger, capable de lecture semantique et de decision autonome pour les cas routiniers. Ne traite pas de problemes nouveaux ni de raisonnement multi-etapes complexe. Escalade vers le Tier 3 avec contexte quand la situation depasse sa capacite.

Tier 3 -- Emperor : raisonnement complet pour les situations genuinement nouvelles. Composite de composants de raisonnement et d'action. Enregistre toutes ses decisions, chemins de raisonnement et resultats -- ce corpus alimente l'apprentissage des tiers inferieurs.

Le pipeline est bidirectionnel. L'escalade ascendante (upward escalation) achemine les operations vers un tier de capacite superieure. L'optimisation descendante (downward optimisation) fait migrer les patterns appris par un tier superieur vers des tiers inferieurs plus rapides.

L'arc d'apprentissage se deploye en quatre phases :

1. **Phase 1 -- Emperor Only** : toutes les operations atteignent l'Emperor. Phase couteuse et intentionnellement ainsi -- elle construit le corpus d'apprentissage. "The Emperor's deliberate, expensive processing in early phases is not inefficiency -- it is the investment that makes future efficiency possible."
2. **Phase 2 -- DTR Active** : le DTR est entraine sur le corpus de la Phase 1. Les patterns deterministes repetes sont routes directement.
3. **Phase 3 -- Executor Active** : l'Executor prend en charge les operations trop semantiques pour le DTR mais trop routinieres pour l'Emperor.
4. **Etat mature** : le DTR traite la majorite, l'Executor traite le reste, l'Emperor ne voit que ce qu'aucun tier inferieur ne peut resoudre. Le systeme est significativement plus rapide, moins couteux et plus efficace qu'en Phase 1.

L'ingenierie du contexte (context engineering) n'est pas un tier du PCP. Elle opere en parallele pour assembler un contexte sur mesure destine a l'Emperor.

La specialisation de domaine (domain specialisation) est dans les donnees, pas dans l'architecture. L'algorithme DTR, le mecanisme Executor et la classe Emperor sont identiques a travers tous les deployments. Ce qui differe : les donnees d'entrainement propres a chaque domaine.

3.3 Le Decision Tree Router en detail

Le paper D3 ("Decision Tree Router") descend dans la mecanique interne du DTR. Le composant est un classifieur non-LLM positionne en entree de pipeline. Son algorithme est un Hoeffding tree (arbre de Hoeffding, aussi appele Very Fast Decision Tree / VFDT), choisi pour sa capacite d'apprentissage en ligne sur flux continu sans re-entrainement batch.

Le flux de traitement suit un ordre strict :

1. **Contraintes constitutionnelles** (constitutional constraints) : regles hard-codees evaluees avant toute extraction de features. Immuables, incontournables par la classification. DENY ou ALLOW absolus.
2. **Extraction du vecteur de features** (feature vector) : representation compacte de 25-40 dimensions -- champs structures, presence de mots-cles, marqueurs de contenu, signaux de contexte conversationnel. Cible : extraction en moins de 10 microsecondes.
3. **Classification par le Hoeffding tree** : produit une classification (ALLOW, DENY, DEFER) et un score de confiance.
4. **Application du seuil de confiance** (confidence threshold) : $\geq 0,90$ pour une decision deterministe. $< 0,90$ implique DEFER, quel que soit le resultat nominal.
5. **Execution ou escalade** : ALLOW (execution directe), DENY (rejet immediat), DEFER (escalade a l'agent LLM).
6. **Boucle de feedback** : apres chaque requete, le DTR observe l'outcome et met a jour son arbre. Un routage correct renforce le chemin ; un routage incorrect l'affaiblit. Les requetes escaladees et approuvees par l'agent sont marquees comme candidates pour un futur fast-path.

"The correctness of the system does not depend on the DTR. It depends on the agent. The DTR is a performance and efficiency optimization, not a correctness component."

Propriete de securite fondamentale : si le DTR est mal configure ou rencontre un pattern inconnu, le fallback est toujours l'escalade vers l'agent -- sans dommage. La correction du systeme ne depend pas du DTR.

Les cibles de performance sont des design targets (le DTR est "Designed, Not Yet Built") : latence de 10-100 microsecondes, debit de 10 000-100 000 classifications par seconde, empreinte memoire < 10 MB, utilisation CPU $< 1\%$, aucun GPU requis.

Chaque instance DTR est co-localisee avec son pipeline et entrainee a partir de l'historique operationnel de ce pipeline. L'algorithme est partage ; les donnees et le modele resultant sont propres a chaque domaine.

3.4 L'architecture code agent-optimale

Le paper D4 ("Agent-Optimal Code Architecture") pose que le code destine a etre construit et maintenu par des agents doit adopter une architecture fondamentalement differente. Deux principes : organiser toute la logique applicative sous forme de StateGraphs, et separer strictement l'infrastructure (AE) de l'intelligence (TE).

Le StateGraph est l' unite fondamentale. Chaque noeud est une fonction a usage unique. Les aretes rendent le flux de controle explicite. L'etat est le seul element qui circule entre les noeuds. Le graphe est simultanement documentation, specification et implementation.

"The StateGraph is not an orchestration layer on top of application code. It IS the application."

L'invariant tout-logique-dans-le-graphe (All-Logic-In-Graph Invariant) stipule que tout code logique doit resider dans un StateGraph. Le code hors-graphe n'est tolere que pour le cablage de transport ou la mecanique triviale. Le raisonnement : quand un agent lit du code procedural, il doit reconstruire l'intention depuis les details d'implementation -- inferer le flux de controle depuis les conditionnelles, inferer l'etat depuis les assignations, inferer le but depuis le nommage. Cette reconstruction est sujette aux erreurs. Le StateGraph elimine cette reconstruction en rendant tout explicite.

Capacites natives du StateGraph : les cycles (boucles de polling, raffinement iteratif), la persistance (checkpoint entre executions, survie aux redemarrages), la composabilite (un noeud peut invoquer un autre graphe compile).

La separation AE/TE est la consequence architecturale majeure. L'AE (conteneurs, passerelle, pools de connexions, BDD, observabilite) est stable par nature et change rarement. Le TE (Imperator, PCP, flux d'inference, raisonnement domaine) change constamment. Quand les deux sont separes, chacun evolue a son propre rythme.

L'intelligence devient un package portable et versionnable. Le conteneur se reduit a un bootstrap kernel (noyau d'amorçage) -- runtime, serveur web, gestion des connexions, et unique capacite : charger des packages. Ce qui definit l'identite d'un MAD, ce sont ses packages, non ses conteneurs. Le conteneur est infrastructure generique ; les packages sont l'identite.

L'arc d'evolution est decrit en cinq etats :

- **State 0 -- Monolithique** : tout dans une application indifferenciee.
- **State 1 -- Structure** : structure interne visible mais deploiement monolithique.
- **State 2 -- StateGraph valide** : invariant graphe etabli, les agents peuvent raisonner sur le systeme.

- **State 3 -- Pleinement separe** : TE et AE deviennent des packages independants, le cycle cognitif est ecrire-publier-installer.
- **State 4 -- Dependances configurables** (configurable dependencies) : chaque dependance externe devient un choix de configuration. `docker compose up` sur n'importe quelle machine. Le code ne change jamais entre environnements -- seule la configuration change.

L'eMAD n'est possible qu'avec un TE portable : creer un nouvel agent devient economiquement trivial -- pas d'infrastructure a provisionner, juste un package a publier. En State 4, la barriere a l'evaluation passe de "configurer l'ecosysteme" a "docker compose up". Le Metacognitive Architect (Architecte Metacognitif) commissionne de nouvelles capacites en publiant des packages -- aucun humain ne touche un conteneur.

3.5 Le modele de communication inter-agents

Le paper D5 ("Agent Communication Model") detaille comment les MADs communiquent dans l'ecosysteme. Le modele repose sur trois separations : stockage du message (Conversation Broker), notification (trigger leger), et decision de participation (Moderator Executor, executeur modérateur).

Le modele d'adressage (addressing model) reprend le schema From/To/CC de l'email. La semantique To vs CC est une distinction d'obligation : To impose une reponse, CC est une invitation facultative. Les participants a une conversation sont derives de l'historique des messages, pas declares a l'avance.

"Participants are an artifact of the message, not the conversation."

Le garde-fou structurel primaire contre les boucles en cascade est le no reply-to-all (pas de repondre-a-tous) : toute reponse s'adresse par default au seul expéditeur original. La liste CC n'est jamais heritee automatiquement.

Le flux de communication complet passe par sept etapes :

1. **Composition et adressage** par le Sending Moderator (modérateur d'envoi).
2. **Stockage central** dans le Conversation Broker avant toute notification.
3. **Protection anti-boucle** : hop counting (comptage de sauts, emprunte au modele Received de l'email), rate limiting, deduplication.
4. **Declenchement** : trigger leger portant uniquement le conversation_id. L'agent lit le contenu depuis le Conversation Broker.
5. **Reception et routage** par le Receiving Moderator (modérateur de reception) : messages To routes vers le handler obligatoire, messages CC soumis a deliberation.

6. **Niveaux de routage internes** : Tier 1 (regles programmatiques), Tier 2 (inference ~7B), Tier 3 (DTR appris).

7. **Cycle de curation** (curation cycle) : log immediat avec statut pending_evaluation, puis evaluation differee en arriere-plan (signaux immediats : boucles, echecs ; signaux court terme : contributions ; signaux long terme : objectif atteint). Resultat : score de qualite, promotion dans le graphe de connaissance, exemple d'entrainement pour le DTR.

Le graphe de connaissance des decisions de moderation (Moderation Decision Knowledge Graph) est une structure a trois dimensions (Trigger, Decision, Quality) stockee dans le Context Broker. Il fournit une memoire institutionnelle des patterns de communication. Il ne duplique pas le contenu des messages -- il capture les decisions en abstrait.

La maturite progressive du DTR de moderation suit les memes trois phases que le PCP general : precocite (Imperator traite la plupart), en developpement (DTR commence a router), mature (DTR route la majorite).

3.6 Le paquet d'ancrage

Le paper D6 ("The Anchor Package") traite du probleme d'alignement initial dans les taches multi-participants. L'Anchor Package (paquet d'ancrage) est un bloc consolide de donnees -- documents, instructions, contexte, materiel de reference -- assemble specifiquement pour aligner tous les participants avant et pendant l'execution. Sa propriete definissante est l'uniformite (uniformity) : chaque destinataire recoit le paquet identique, sans distribution partielle ni personnalisation.

"The Anchor Package solved this by separating the assembly of the shared starting point from the generation work itself. Assemble once; distribute identically; generate independently."

Le concept est ne lors de la conception du pattern Quorum, ou la diversite cognitive des modeles posait un probleme : des modeles generant independamment doivent partir du meme point de depart exact, sinon leurs outputs ne sont pas comparables.

Le contenu typique inclut : documents de reference ecosysteme (HLD, REQ), specification de tache, travaux anterieurs pertinents, instructions de travail, definitions de role/persona, memoire recuperee depuis le Context Broker. L'assemblage peut etre schema-driven (pilote par schema, via StateGraph), reason-driven (pilote par raisonnement, un agent selectionne les composants) ou hybride (cas commun : StateGraph pour les portions fixes, agent pour les portions specifiques a la tache).

La separation des responsabilites est explicite : le Context Broker peut fournir des composants (historique, faits extraits, decisions), mais il ne connait pas ce qui est ancre, qui recevra le paquet, ni ce dont les participants ont besoin. Cette comprehension appartient a l'orchestrateur ou au processus propriétaire de la tache.

Cinq cas d'usage sont identifiés : Quorum pattern (le plus structure), deploiement d'agents en parallele, hydratation d'agent ephemere (ephemeral agent hydration -- initialiser un nouvel agent sans contexte prior), ancrage de conversation (integrer un nouveau participant en cours de discussion), et continuite inter-sessions (cross-session continuity -- taches longues traversant plusieurs cycles de vie d'agents).

Le paper inclut un cas d'etude empirique (Case Study C03, Blueprint v2.0.2) : un transcript vocal de 25 minutes (3 918 mots verbatim) preserves intact comme ancre pour toutes les phases de developpement. Les reviewers LLM ont mesure une fidelite de 85 a 98% a l'intention originale. L'approche verbatim-as-anchor reduit substantiellement la degradation "effet telephone" (telephone game degradation) qui survient quand les exigences passent par plusieurs couches de resume.

3.7 La gestion du filesystem par intention

Le paper D7 ("Intent-Based Filesystem Management") est l'instanciation la plus concrete de la gouvernance par intention definie en A3. Il pose que le controle d'accès traditionnel fonde sur les permissions POSIX est structurellement inadapté aux écosystèmes multi-agents : il ne peut ni exprimer des politiques sémantiques, ni s'adapter à la vitesse et à la nouveauté des opérations à l'échelle machine.

Le modele remplace la question "cet agent a-t-il le droit d'accéder a ce chemin ?" par "comment l'intention de la politique gouverne-t-elle la decision, compte tenu de l'intention de l'action ?"

"The solution is to manage at the level of principles, not rules. Instead of defining every permitted and forbidden operation explicitly, the administrator defines the intent of the governance: what the system is trying to protect, what it is trying to allow, and why."

Toute operation de mutation soumet obligatoirement un parametre intent. Sans ce parametre, l'operation est invalide. L'architecture de decision est a deux niveaux :

- **Tier 1 -- DTR** : chemin rapide pour les operations routinieres, sans invocation LLM.
- **Tier 2 -- Policy Enforcement Agent** (agent d'application des politiques) : modele LLM petit et rapide pour l'evaluation deliberative des cas non resolus par le fast path.

Les outcomes possibles ne sont pas binaires : approbation, refus, approbation avec suggestion corrective (guidance), ou defere pour raisonnement supplementaire.

Le concept de Policy as Conversation (politique comme conversation) est l'evolution de Policy as Code (politique comme code) : les politiques sont exprimees, ajoutees et revoquees en langage naturel sans modification de fichiers de configuration ni redemarrage de service. L'administrateur peut etre une personne, un LLM, ou un Quorum.

La boucle d'apprentissage (learning loop) observe chaque decision et alimente un graphe de connaissances (knowledge graph) modelisant les relations entre roles d'agents, chemins, types d'operations, intentions declarees et decisions passees. Quand un pattern d'approbation repete est detecte, le systeme propose une nouvelle regle fast-path. L'administrateur valide en langage naturel. Le fast path devient plus complet avec le temps.

Les contraintes constitutionnelles (constitutional constraints) sont des interdictions absolues executees avant le routeur, non modifiables par aucune politique ni aucun pattern appris. L'hypothese de cooperation (cooperative assumption) est une limite explicite du modele : les agents sont presumes declarer leur intention honnetement.

L'administrateur devient un architecte de gouvernance (governance architect) : il definit les principes et valide les propositions de la boucle d'apprentissage plutot que de gerer manuellement des permissions individuelles. Le systeme "scale la sagesse humaine a la vitesse machine".

4. Transversalites A<->D

4.1 Les primitives partagees

Plusieurs primitives definies dans le bloc A apparaissent directement dans les architectures du bloc D, formant des ponts structurels entre theorie et implementation.

Le **StateGraph** est le cas le plus net. Introduit dans le vocabulaire du bloc A comme unite de deployment du Thought Engine (A5), il devient dans le bloc D le composant central de l'architecture code (D4), le mecanisme de composition de la boucle ReAct (D1), et l'outil d'assemblage des paquets d'ancrage (D6). Sa capacite de persistance (checkpoint entre executions) est ce qui rend possible l'existence ephemere des eMADs definis en A5. Le StateGraph est le point de convergence le plus concret entre la vision et l'implementation.

L'**intention** (intent) traverse les deux blocs de part en part. Definie comme mecanisme unificateur en A3, elle se materialise dans le DTR comme la ligne distinguant ce qui peut etre route determiniquement de ce qui necessite deliberation (D2-D3), dans le modele de communication comme la semantique qui gouverne le routage des messages (D5), et dans la gestion filesystem comme le parametre obligatoire de toute operation mutante (D7). La gouvernance par intention theorisee en A3 trouve son instantiation la plus complete en D7.

Le pattern **MAD** (A5) se declina dans le bloc D sous forme de contraintes architecturales concretes : la separation AE/TE de D4, le modele de communication de D5, le pattern de deploiement en packages portables. Les deux classes d'agents (Imperator et Executor) definies en A5 structurent directement le PCP (D2) et le modele de communication (D5).

Les **Brokers** (Context Broker, Inference Broker, Conversation Broker) sont introduits dans le bloc A comme infrastructure de gestion de l'etat conversationnel (A2) et de configuration du persona (A6). Dans le bloc D, ils reapparaissent comme fournisseurs de composants pour l'Anchor Package (D6), comme infrastructure de stockage et notification pour le modele de communication (D5), et comme source du contexte ingenie en parallele du PCP (D2).

4.2 La vision architecturale d'ensemble

Vue globalement, l'architecture de Morrisette suit un pattern coherent qui va de l'ontologie a l'implementation en passant par la mecanique.

Le bloc A definit une ontologie agentique : ce qu'est un agent (identite + but + domaine), ce qui constitue son etat (la conversation), ce qui gouverne son action (l'intention), et comment il se compose avec d'autres (le pattern MAD). Ces definitions ne sont pas abstraites au sens philosophique -- elles sont prescriptives, destinees a contraindre directement les decisions de conception.

Le bloc D traduit cette ontologie en mecanismes operationnels. La boucle System 3 ReAct (D1) est la traduction directe du principe "l'agent n'est pas un processus" (A2) : l'agent est un moment de raisonnement entre deux mouvements d'etat conversationnel, et ce raisonnement est decompose en roles cognitifs distincts. Le PCP (D2) operationnalise le principe d'economie cognitive : ne pas gaspiller le raisonnement couteux sur du travail routinier. Le DTR (D3) est la realisation technique la plus detaillee du bloc, descendant jusqu'aux cibles de performance en microsecondes. L'architecture code (D4) traduit le principe d'infrastructure agent-native (A1) en structure de code concret : le StateGraph comme unite fondamentale, la separation AE/TE comme pattern de deploiement.

Le mouvement general est celui d'une descente en resolution : primitives (A1) -> etat conversationnel (A2) -> intention (A3) -> identite (A4) -> pattern MAD (A5) -> persona (A6) -> boucle de raisonnement (D1) -> pipeline cognitif (D2) -> routage deterministe (D3) -> architecture code (D4) -> communication (D5) -> ancrage (D6) -> filesystem (D7).

Cette progressivite n'est pas lineaire au sens strict -- D5 et D6 pourraient etre lus independamment de D3, par exemple -- mais la dependance conceptuelle est generalement descendante : les papers D presupposent les definitions A sans les repeter.

4.3 Tensions et zones floues

Plusieurs tensions apparaissent entre les fondations theoriques et l'architecture proposee.

La conversation infinie et la fenetre de contexte finie. Le modele pose la conversation comme substrat infini de l'etat. Mais la fenetre de contexte de tout LLM est finie. Le paper A2 reconnait cette tension comme un probleme ouvert ("rendre cette conversation utilisable dans des fenetres de contexte finies a grande echelle est un probleme ouvert -- cout computationnel croissant avec le volume") sans la resoudre. Le context engineering est presente comme la discipline qui fait le pont, mais sa mecanique detaillee est renvoyee aux papers du bloc C (Context Broker, Inference Broker) qui ne sont pas couverts ici.

L'hypothese de cooperation et la gouvernance par intention. Le modele D7 repose explicitement sur une hypothese de cooperation (cooperative assumption) : les agents declarent leur intention honnetement. C'est une limite reconnue, mais elle est en tension avec le principe de gouvernance a grande echelle. La gouvernance par intention est presentee comme superieure aux regles POSIX, mais elle suppose la bonne foi du declare -- une propriete que les systemes de securite traditionnels ne presupposent justement pas.

Le statut "Designed, Not Yet Built" du DTR et du PCP. Les papers D2 et D3 sont les plus prescriptifs du bloc D, avec des cibles de performance chiffrees (latence en microsecondes, debit, memoire). Mais ces cibles sont des design targets, pas des mesures. Le DTR n'a pas encore ete construit. Le PCP est decrit en quatre phases dont seule la Phase 1 (Imperator Only) correspond a l'etat actuel. L'arc d'apprentissage complet est une projection, pas un resultat.

La separation trois-modeles de System 3 ReAct. Le paper D1 reconnait explicitement que la separation Reasoner/Actor/Conversation Engine n'est pas implementee : tous les agents Joshua26 utilisent le pattern Imperator (un seul modele). La separation est "la direction architecturale, pas l'etat actuel". La latence supplementaire (jusqu'a trois appels d'inference par iteration) et le cout associe sont reconnus comme contraintes pratiques non resolues.

L'ALM comme horizon speculatif. La vision de modeles de langage agentiques entraines sur la communication machine-a-machine, presente dans les deux blocs (A1 et D1), est explicitement qualifiee de "direction speculative a long terme -- non implementable aujourd'hui". C'est un horizon qui colore la vision d'ensemble mais qui reste au stade de la speculation.

Ces tensions ne sont pas des faiblesses cachees : elles sont, pour la plupart, explicitement reconnues par l'auteur. Le merite du corpus est de les nommer plutot que de les masquer.

5. Limites et positionnement

5.1 Limites du corpus

Le corpus presente plusieurs limites structurelles qu'il convient de reconnaitre.

Source unique. Les 23 papers sont l'oeuvre d'un seul auteur. Il n'y a pas de peer review, pas de co-auteurs, pas de validation externe independante. Le systeme de reference Joshua/Joshua26 est l'experience d'un praticien -- precieuse mais non reproductible par un tiers a partir des informations fournies.

Absence de benchmarks empiriques. Aucun paper des blocs A et D ne contient de mesure experimentale. Les arguments sont de type raisonnement theorique, analogie structurelle et coherence interne. Les six etudes de cas du systeme Joshua sont referencees mais non incluses. Le Case Study C03 de D6 (Blueprint v2.0.2) est le seul element empirique direct, et il porte sur un seul cas d'usage.

Terminologie proprietarie. Le vocabulaire est largement original : MAD, pMAD, eMAD, Imperator, Executor, PCP, DTR, System 3 ReAct, ALM, Anchor Package, etc. Aucun de ces termes n'est utilise dans la litterature existante avec le meme sens. Le risque est une terminologie insulaire, difficile a mettre en correspondance avec les cadres theoriques etablis.

Niveaux de maturite variables. Le paper cadre (A1) est un "Framework Document", le PCP (D2) et le DTR (D3) sont "Designed, Not Yet Built", la conversation comme etat (A2) est "Partially Implemented", l'architecture code (D4) est illustree par une progression en cinq etats dont seuls les premiers sont realises. Le corpus melange des elements implements, partiellement implements et purement conceptuels sans toujours clarifier la frontiere.

5.2 Ce que le corpus apporte

Malgre ces limites, le corpus apporte des contributions identifiabiles.

Un vocabulaire structure pour des concepts diffus. Le champ de l'ingenierie agentique manque de vocabulaire partage. Les notions d'identite agent, de conversation comme etat, de gouvernance par intention existent a l'etat diffus dans les pratiques -- Morrisette leur donne des noms, des definitions et des relations. Le fait que la terminologie soit proprietarie est une faiblesse pour la diffusion mais une force pour la precision interne.

Une architecture prescriptive coherente de bout en bout. Les 13 papers couverts ici forment un systeme ou les primitives (A) se materialisent dans les mecanismes (D) avec une coherence interne rare dans les concept papers. Les references croisees entre papers sont explicites et verifiables.

Un traitement explicite de l'apprentissage en ligne. Le pattern DTR -> Executor -> Imperator, avec sa boucle d'optimisation descendante, propose un modele concret d'amelioration progressive des systemes agentiques sans re-entrainement batch. Ce modele est non teste mais architecturalement complet.

Une separation claire entre identite et persona. La distinction configuration vs. architecture, avec ses implications pour les signaux d'apprentissage a deux vitesses, est une contribution conceptuelle utile pour quiconque conçoit des agents persistants.

5.3 Positionnement dans le champ

Le corpus se situe dans l'espace des architectures multi-agents prescriptives. Il partage des preoccupations avec les frameworks d'orchestration agentique (LangGraph, AutoGen, CrewAI) mais se distingue par son niveau d'abstraction : la ou ces frameworks fournissent des outils, Morrisette propose un cadre conceptuel complet.

L'approche est plus proche des position papers en architecture logicielle que des publications empiriques en IA. Le format concept paper series est inhabituel dans le champ : 23 documents formant un argument progressif, publies sur une plateforme non-academique, sans les contraintes de structure et de review des conferences.

La vision d'ensemble -- un ecosysteme de specialistes autonomes communiquant par protocole semantique, apprenant de leur propre experience, avec une metacognition recursive -- est radicalement ambitieuse. Elle ne sera validee ou invalidee que par l'implementation et la mesure.

6. Annexe -- Index des papers couverts

#	ID	Titre	Bloc	Statut maturite
1	A1	Con conversationally Cognizant AI Systems	A - Foundational	Framework Document
2	A2	Conversation as State	A - Foundational	Partially Implemented
3	A3	Intent	A - Foundational	Concept / Implementable
4	A4	Agent Purpose and Identity	A - Foundational	Concept / Implementable
5	A5	The Multipurpose Agentic Duo	A - Foundational	Partially Implemented
6	A6	Agentic Identity and Persona	A - Foundational	Concept / Implementable
7	D1	System 3 ReAct	D - Architecture	Designed, Not Yet Built
8	D2	Progressive Cognitive Pipeline	D - Architecture	Designed, Not Yet Built
9	D3	Decision Tree Router	D - Architecture	Designed, Not Yet Built
10	D4	Agent-Optimal Code Architecture	D - Architecture	Partially Implemented
11	D5	Agent Communication Model	D - Architecture	Designed / Partially Implemented
12	D6	The Anchor Package	D - Architecture	Partially Implemented
13	D7	Intent-Based Filesystem Management	D - Architecture	Designed, Not Yet Built

7. Sources

- Repo GitHub : <https://github.com/jmorrisette-RMDC/portfolio/tree/main/Joshua26/docs/concepts/>
 - Substack : jmorrisettermdc.substack.com
 - Serie : "Conversationally Cognizant AI", mars 2026
 - Extractions structurees produites en S90 ([data/S90/extractions-morrisette/](https://data.s90.com/extractions-morrisette/))
-