Raphaël Nguyen
Raphaël Chauvier

Aym

Assemble Your Memories

Une expérience de conversation guidée et visuelle pour explorer ses propres idées

---

Rapport de projet

SI28 : écriture interactive et multimédia

Université de Technologie de Compiègne

Semestre A25

Table des matières

1 Introduction

1.1 But du projet en une phrase

En une phrase, *Assemble Your Memories (AYM)* est une petite expérience web où l'on discute avec un agent conversationnel qui, au lieu de simplement donner des conseils, nous invite à construire un **graphe interactif de nos propres stratégies et conséquences futures attendues** pour nous aider à comprendre nos attentes et à réfléchir à nos options.

1.2 AYM dans le contexte de SI28

Le cours SI28 met l'accent sur la création de dispositifs numériques où **l'interaction** et **l'écriture** sont intimement liées : récits hypertextuels, expériences participatives, jeux vidéo narratifs, installations d'art numérique, etc. AYM s'inscrit dans cette continuité, mais avec un angle un peu différent :

— l'enjeu n'est pas de raconter une histoire de fiction, mais de **mettre en scène le propre récit de l'utilisateur** sur son futur, au travers du graph et de la conversation, de manière à provoquer en lui certaines réactions ;

— l'expérience veut pousser l'utilisateur à *s'exprimer*, en reproduisant une forme de conversation naturelle avec un interlocuteur "vivant" et perspicace qui questionne l'utilisateur sur ses décisions, ses doutes, ses projets.

— au fil de l'expérience, l'utilisateur crée un *objet numérique unique*, le graph, qui peut être consulté restrospectivement et partagé.

Nous avons donc choisi de traiter le LLM et le graphe *comme un dispositif d'écriture interactive* littéral.

1.3 Le défi du scénario : de l'écriture linéaire à l'écriture de règles

Dans une acception classique du multimédia (type CD-Rom ou webdocumentaire), le scénario contient « l'ensemble des textes qui vont apparaître à l'écran ». Avec l'usage d'un LLM, cette définition doit évoluer : il est impossible de rédiger à l'avance les réponses exactes de l'agent.

Cependant, nous défendons l'idée qu'AYM repose sur une **scénarisation forte**, mais déplacée du *contenu* vers la *structure* :

— **Le Prompt System comme scénario** : Au lieu d'écrire les dialogues, nous écrivons les *contraintes* psychologiques et les *objectifs* de la scène. Le "System Prompt" agit comme un metteur en scène qui dirige un acteur d'improvisation. C'est une forme d'écriture indirecte mais rigoureuse.

— **Les Milestones (Jalons) comme nœuds narratifs** : Pour éviter une expérience totalement libre et informe, nous avons intégré des *Milestones* (objectifs ludiques et narratifs). L'utilisateur ne "finit" pas l'expérience tant que certains états du graphe ne sont pas atteints. Cela permet d'avoir un certain contrôle sur la *trajectoire de l'expérience*, d'encourager l'utilisateur à expérimenter ceci ou cela. (Notez que les milestones sont encore à un état expérimental dans notre projet)

Ainsi, AYM propose un **scénario probabiliste** : nous ne savons pas quels mots seront utilisés, mais nous avons scénarisé les étapes de réflexion par lesquelles l'utilisateur doit passer, et différents aspects de son expérience via le comportement de l'agent et les choix d'interface.

1.4 Organisation du rapport

Ce rapport suit la structure suivante :

— **Section 2** : positionnement, intentions et inspirations (en lien avec le cours SI28) ;

— **Section 3** : description des écrans, des menus et de l'expérience utilisateur ;

— **Section 4** : fonctionnement interne de l'agent conversationnel (prompts, orchestration, étapes de traitement) ;
— **Section 5** : structure du graphe et manière dont le LLM l'utilise ;
— **Section 6** : aspects pratiques (choix de modèles, historique, hébergement, multi-utilisateurs) ;
— **Section 7** : conclusion, difficultés rencontrées et pistes d'amélioration.

2 Positionnement, intentions et inspirations

2.1 Une expérience interactive à vivre plutôt qu'un service

Concrètement, AYM propose à l'utilisateur de s'exprimer sur un sujet qui le préoccupe. Plutôt qu'à lui fournir des informations clés en main, l'agent cherche à lui poser des questions qui l'amènent à réfléchir, à explorer des points de tension. Cela donne :
— un échange court et dynamique, qui explore naturellement un grand nombre de sujets ;
— une carte visuelle de stratégies possibles et de conséquences anticipées produites par l'utilisateur ;

On est donc plus proche d'un **dispositif d'écriture de soi** que d'un service d'aide à la décision :
— l'objectif n'est pas que l'IA « trouve la bonne solution » ;
— l'objectif est que l'utilisateur voie mieux ce qu'il pense déjà, ce qu'il suppose, ce qu'il n'a pas encore exploré.

Cette posture rejoint directement les préoccupations de SI28 :
— comment un dispositif numérique peut-il transformer la manière dont on écrit (ici : sur soi, son avenir) ?
— quelles formes interactives peuvent aider à **prendre du recul** sur ses propres récits ?
— que se passe-t-il quand on donne une « forme graphique » à ce qu'on raconte ?

2.2 Inspirations : des chatbots à la maïeutique visuelle

Rappels historiques : d'ELIZA aux agents outillés

Une première source d'inspiration évidente est **ELIZA**, le programme de Joseph Weizenbaum qui simulait un pseudo-psychanalyste en reformulant les paroles de l'utilisateur. AYM s'en rapproche sur plusieurs points :
— la conversation est centrée sur la personne qui écrit ;
— l'agent ne donne pas de conseils directs mais renvoie des questions ;
— l'effet recherché est que l'utilisateur parle de lui-même et réfléchisse.

Mais il y a une différence majeure :
— ELIZA se contentait de **reformuler** sans mémoire structurée ;
— AYM, lui, **matérialise** progressivement ce qui se dit dans un graphe explicite de stratégies et de prédictions.

Plus largement, on s'inscrit dans la vague récente des **agents LLM outillés** :
— un LLM ne répond pas seulement en texte libre, il manipule aussi des structures de données ;
— il peut appeler des « outils » (ici un outil de mutation de graphe, un outil d'ouverture d'entrée utilisateur, etc.) ;
— il suit un **plan de conversation** qui est lui-même explicité et renvoyé à l'interface.

Graphes, cartes mentales et réflexion guidée

Une autre source d'inspiration vient des cartes conceptuelles, des mind-maps et des outils de prise de notes structurées. Beaucoup de personnes disent qu'elles y voient plus clair quand

elles :

— dessinent un schéma sur papier ;
— mettent des flèches entre des idées ;
— ou utilisent un logiciel de carte mentale.

AYM reprend cette intuition mais essaye de **l'automatiser par la conversation** :

— l'utilisateur n'a pas besoin de « penser en graphes » ;
— il parle normalement, et l'agent se charge de détecter dans ses phrases des *stratégies* (« je pourrais faire X ») et des *prédictions* (« je m'attends à Y ») ;
— ces éléments sont alors ajoutés au graphe avec des liens logiques simples (Stratégie → Prédiction, Prédiction → Stratégie, Prédiction → Prédiction).

On retrouve ici notre intuition de départ : *un graphe peut être une bonne externalisation de ce qu'on a en tête, mais ce n'est pas forcément naturel de le dessiner soi-même.* C'est cette partie « pénible » que l'on délègue au LLM.

Parler à quelqu'un pour y voir plus clair

Un autre point de départ, plus quotidien, vient de l'expérience très banale suivante :

— parfois, le simple fait de raconter un problème à un ami ou à un psy permet d'y voir plus clair ;
— la personne en face n'a pas besoin d'avoir la solution ;
— le fait de **devoir formuler**, **répondre à des questions**, **revenir sur certains points** suffit déjà à réorganiser le problème.

C'est un peu cette idée que nous avons voulu retrouver ici :

— l'agent ne se présente pas comme un expert, mais comme un « ami stratégique » qui pose des questions ;
— les questions cherchent à faire émerger des options (« Et si tu faisais ça ? ») et des conséquences (« Et dans ce cas, tu t'attendrais à quoi ? ») ;
— le graphe joue le rôle de « cahier de notes » qu'on aurait rempli à plusieurs, mais également de *contrainte*, qui encadre la manière d'y ajouter ses idées.

Éviter les réponses toutes faites et les biais commerciaux

Une autre motivation derrière AYM est de **limiter l'effet « boîte noire »** du LLM qui donne des réponses toutes faites, parfois biaisées, parfois obsolètes :

— dans beaucoup d'applications, le modèle est utilisé comme une encyclopédie universelle qui renvoie des faits ou des conseils ;
— cela met l'utilisateur dans une posture assez passive : il « consomme » une réponse.

Dans AYM, on essaie de renverser ce rapport :

— le LLM sert surtout à générer des **questions** et des **pistes de réflexion**, pas des prescriptions ;
— l'utilisateur reste celui qui *définit* ses propres stratégies et ses propres prédictions ;
— le graphe est construit à partir de ce que l'utilisateur a dit, pas à partir d'une base de connaissances externe.

Autrement dit, on peut « faire réfléchir les gens par eux-mêmes », les encourager à partager des informations et à se regarder avec un œil critique, sans les submerger de recommandations issues d'un modèle entraîné sur des données potentiellement commerciales, datées ou partiales.

Vers une communauté de graphes vivants ?

Enfin, derrière AYM, on a aussi une idée plus spéculative :

— si plusieurs personnes utilisaient ce type d'agent sur un même thème (par exemple un problème mécanique, un sujet de santé, un même type de projet), on pourrait imaginer mutualiser leurs graphes ;

— à la longue, ça pourrait devenir une **source d'information** construite *par les usages* : des stratégies testées, des conséquences observées, des objectifs récurrents.

Pour le projet SI28, nous sommes restés au niveau d'une expérience individuelle. Mais le design du graphe et des outils de mutation laisse ouverte cette possibilité de **passer à l'échelle communautaire** plus tard :

— les nœuds sont assez génériques (labels courts, contenus textuels détaillés) ;
— les liens Stratégie/Prédiction sont simples ;
— la structure est facilement sérialisable et partageable.

2.3 Public cible et types d'usages

Qui a envie d'utiliser AYM ?

Concrètement, on imagine AYM comme un outil pour des personnes qui :

— sont face à une décision ou à un problème un peu complexe (orientation, projet, relationnel, organisation de travail, etc.) ;
— aiment le format conversationnel (un chat) et ne veulent pas passer par un gros logiciel de mind-mapping ;
— ont envie de *prendre un temps pour réfléchir* sans forcément en parler à quelqu'un de proche.

Dans le cadre du cours, le premier public évident, ce sont les **étudiants** eux-mêmes :

— très concrètement, on peut utiliser AYM pour réfléchir à son semestre, à un projet de stage, à une décision entre deux cursus ;
— ça peut aussi être un prétexte pour explorer la psychologie du personnage « IA » avec lequel on discute.

Mais on peut très bien imaginer :

— des personnes en reconversion professionnelle ;
— des créateurs (graphistes, devs, musiciens) qui cherchent à structurer un projet ;
— voire des usages plus quotidiens (organiser un déménagement, préparer un voyage, anticiper un conflit).

Situations d'usage typiques

Un scénario d'usage courant pourrait ressembler à ceci :

a) **Onboarding** : la personne arrive, entre son prénom, choisit le prénom de l'agent, téléverse une image de son « ami numérique ».
b) **Choix du sujet** : l'agent demande simplement « Quel sujet ou problème veux-tu explorer aujourd'hui ? ». L'utilisateur parle d'un projet, d'une hésitation, d'un souci récurrent.
c) **Exploration guidée** : l'agent pose des questions ouvertes, propose quelques pistes (« Et si tu faisais plutôt X ? ») et essaye à chaque fois de faire expliciter ce que l'utilisateur s'attend à voir comme résultat.
d) **Construction du graphe** : pendant ce temps, le graphe évolue : de nouveaux nœuds apparaissent, les liens se densifient, certaines zones restent vides.
e) **Prise de recul** : au bout d'un moment, l'utilisateur peut zoomer dans le graphe, tourner la caméra, voir quelles stratégies il a accumulées, quels objectifs ressortent le plus.
f) **Clôture** : la session se termine naturellement quand l'utilisateur sent qu'il a « fait le tour ». Le graphe reste une trace de ce moment de réflexion. Le système de *milestones* peut également lui envoyer un signal de fin (lorsqu'il a atteint tous les objectifs "ludiques").

3 Interface, écrans et expérience utilisateur

3.1 États de l'application

L'application Streamlit est structurée autour de trois grands états d'interface, gérés par une variable `stage` dans l'état global (`AppState`) :

a) **Onboarding** (`"onboarding"`) : paramétrage initial (noms, image de l'agent, options d'expressions faciales) ;

b) **Préchargement des portraits** (`"loading"`) : génération en tâche de fond des variantes d'image (« réflexion », « dessin ») à partir du portrait de base ;

c) **État principal** (`"main"`) : l'écran de conversation avec le graphe et la carte de l'agent.

Cette structuration simple nous a permis d'avoir une **progression claire** dans l'expérience :
— on commence par préparer le dispositif (onboarding) ;
— on laisse l'application faire un peu de travail caché (preload d'images) ;
— puis on entre dans le cœur de l'expérience (conversation + graphe).

3.2 Écran d'onboarding : appropriation de l'agent

L'écran d'onboarding a deux objectifs :
— **technique** : récupérer un portrait de référence pour l'agent, afin de générer ensuite des variations d'expressions ;
— **narratif** : permettre à l'utilisateur de *s'approprier* l'agent en le nommant et en choisissant son apparence.

Concrètement, cet écran contient :
— un champ texte « Ton nom » ;
— un champ texte « Nom de ton ami » (le prénom de l'agent) ;
— un *file uploader* pour téléverser une image (png/jpg) de l'ami ;
— si un portrait existe déjà dans le dossier, une option « Réutiliser le dernier portrait importé » avec prévisualisation ;
— une case à cocher « Désactiver les expressions faciales (Image fixe) ».

Ce dernier point est important pour l'aspect pratique :
— sur certaines machines, on ne dispose pas d'API de génération d'images ;
— ou bien on préfère éviter des requêtes supplémentaires ;
— la case permet donc de forcer l'utilisation d'une image neutre unique.

Le bouton « Confirm » valide l'onboarding. Si aucun portrait n'est fourni ni réutilisé, l'application affiche un avertissement et bloque la suite, ce qui évite d'entrer dans la conversation sans visage pour l'agent.

3.3 Écran principal : chat, carte de l'agent et graphe

L'écran principal est divisé en deux colonnes principales, plus une grande zone en dessous.

Colonne de gauche : la conversation

La colonne de gauche est centrée sur la **conversation** :
— en haut, on a un conteneur « Conversation » qui affiche l'historique des messages ;
— les messages de l'agent (rôle `ai`) sont associés au portrait courant de l'agent ;
— les messages spéciaux (résumé, graph) sont différenciés par leur rôle ;
— sous le flux de messages, un champ `st.chat_input` permet à l'utilisateur d'envoyer un nouveau message.

Un détail important est que ce champ est **désactivé** tant que l'agent n'a pas « relâché » la main. Concrètement :

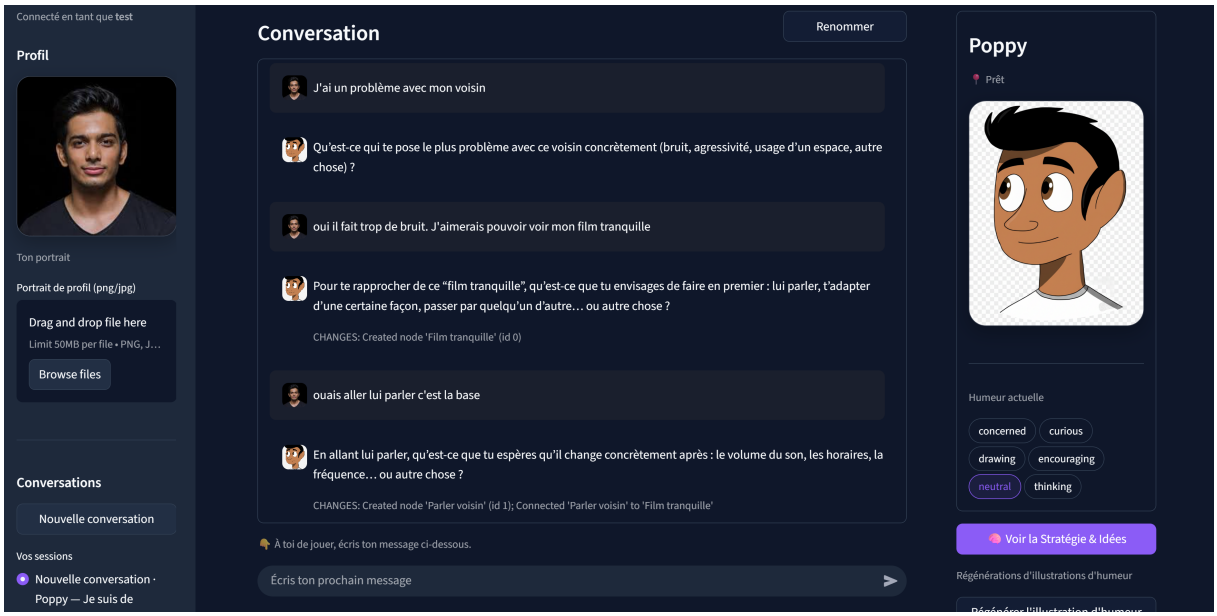

FIGURE 1 – Vue d'ensemble de l'interface principale : le flux de conversation à gauche et la carte de l'agent (portrait et statut) à droite, créant un face-à-face numérique.

— le flux d'événements internes (`WaitEvent`, `FlagEvent`) indique à Streamlit quand l'agent « réfléchit » ou « dessine » ;
— pendant ce temps, le bouton d'envoi reste gris, accompagné d'un petit statut (« Ton ami réfléchit. ») ;
— dès que l'agent est prêt, le statut passe à « À toi de jouer » et le champ est réactivé.

Cela donne un rythme plus humain à l'échange, au lieu d'avoir un modèle qui répondrait instantanément à chaque frappe.

Colonne de droite : carte de l'agent et stratégie

La colonne de droite contient la **carte de l'agent** :
— le nom de l'agent ;
— le portrait courant (expression faciale associée à l'état interne) ;
— un statut textuel : « prêt pour toi », « en réflexion », « dessine pour toi » ;
— une série de « chips » affichant les noms des moods déjà générés (joyeux, sérieux, curieux...).

Juste en dessous, un panneau pliable « Stratégie & Idées » affiche la **stratégie courante de l'agent**, telle que renvoyée par l'LLM :
— c'est un texte qui synthétise le plan de conversation (par exemple : « 1. Clarifier ton objectif principal. 2. Explorer deux options concrètes. 3. Évaluer les conséquences de chaque option. ») ;
— il rappelle aussi les grandes lignes du « document d'idées » utilisé en interne.

Cela permet à l'utilisateur de **voir le raisonnement méta** de l'agent :
— on n'est pas seulement face à des questions isolées ;
— on voit le fil rouge, ce qui donne plus de transparence au dispositif.

Enfin, la colonne propose :
— un bouton (à terme) pour régénérer manuellement les images de mood ;
— un bouton « Reset graph » qui vide complètement le graphe (nœuds, arêtes, jalons) et le remet à zéro.

Zone inférieure : graphe 3D et objectifs

Sous les deux colonnes, une zone pleine largeur affiche :

a) **la visualisation du graphe** : un graphe 3D Plotly, manipulable à la souris (zoom, rotation, déplacement), où chaque nœud est un élément de stratégie ou de prédiction ;

b) **un tableau des objectifs (milestones)** : une table qui liste plusieurs objectifs ludiques (par exemple « Bananes », « Remplisseur de graphe », etc.) avec un pourcentage de progression.

Pour le graphe :

— les nœuds récents peuvent être mis en évidence (couleur différente, indicateur « new ») ;
— les arêtes représentent les liens cause-conséquence ;
— un survol permet d'afficher un tooltip avec le contenu complet du nœud.

L'utilisateur peut ainsi **suivre l'évolution de la carte** au fil de la conversation :

— on voit des branches se développer ;
— on peut repérer des zones encore vides ;
— on voit quel type de stratégies revient souvent.

3.4 Dimension multimédia : images génératives et graphe spatial

Le cours SI28 insiste sur l'association Texte / Image / Son / Vidéo. Si AYM est une expérience web, elle dépasse le simple affichage de texte par une utilisation dynamique du média visuel.

L'image dynamique : vers une forme d'animation procédurale

Un élément assez spécifique d'AYM est la gestion des **expressions faciales** de l'agent :

— le LLM principal renvoie à chaque tour un champ `facial_expression` (par exemple « curious », « thoughtful », « expecting ») ;
— la passerelle d'images (`ImageGateway`) se charge alors de trouver ou de générer une image correspondant à cette humeur ;
— cette image est affichée dans la carte de l'agent et dans certains états d'attente.

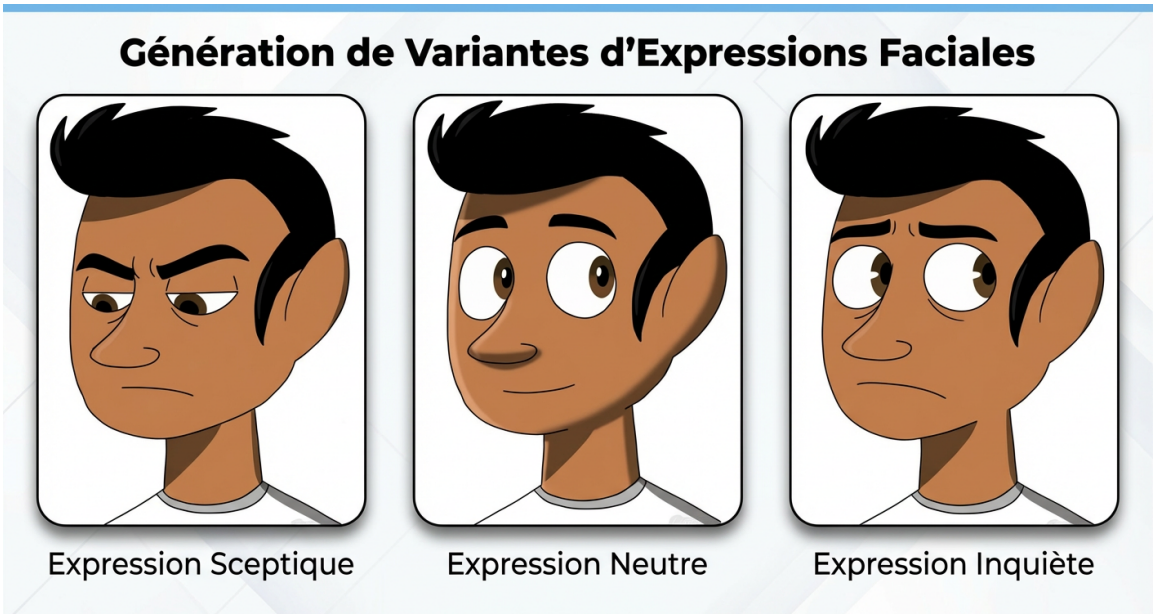

FIGURE 2 – Génération dynamique d'expressions : à partir d'un portrait initial, l'agent adopte visuellement l'attitude (curiosité, réflexion, joie) dictée par le contexte de la conversation.

Au-delà du simple côté « joli », ces expressions faciales ont plusieurs effets intéressants :

— **Lien affectif** : Elles peuvent aider l'utilisateur à humaniser l'agent. Un visage qui change d'humeur donne l'impression d'une présence plus incarnée ;
— **Sentiment d'Immersion** : Il est renforcé car l'expression faciale s'accorde avec le ton actuel de la conversation ;
— **Retour visuel immédiat** : L'utilisateur ne *lit* pas seulement que l'agent réfléchit, il le *voit* changer d'attitude visuelle. Le portrait en « mode réflexion » ou en « mode dessin » donne un feedback clair sur ce que fait la machine.

Pour rester réalistes dans le cadre du projet, on a gardé la possibilité de **désactiver** ces moods :

— certains environnements n'ont pas accès aux API d'images ;
— sur une simple machine de démo, il est parfois plus confortable de n'afficher qu'une seule image.

Bien qu'il ne s'agisse pas de vidéo pré-calculée, cette succession d'états visuels (Neutral → Thinking → Drawing → Happy) crée une **narration visuelle** animée qui compense l'absence de pistes vidéo lourdes. L'image devient un vecteur d'information temps-réel au même titre que le texte.

Le graphe 3D comme média visuel

Nous considérons également le **graphe 3D interactif** non pas comme une simple "dataviz", mais comme un média à part entière :

— Il spatialise la pensée : l'utilisateur manipule des concepts dans un espace 3D (rotation, zoom).
— Il raconte l'histoire de la session : la structure qui se dessine est la *trace visuelle* de la conversation.

L'association Texte (Chat) / Image (Avatar dynamique) / Espace 3D (Graphe) constitue donc le cœur multimédia du projet.

3.5 Choix techniques d'interface (sans rentrer dans le code)

D'un point de vue technique, quelques choix importants ont orienté le design :

— **Streamlit** comme framework web : cela nous a permis de construire rapidement un prototype interactif, avec des composants de chat et des graphiques intégrés, sans gérer manuellement tout le HTML/CSS/JS ;
— **Plotly** pour la 3D : on profite de la capacité de zoom, rotation, survol, simplement en renvoyant une figure 3D depuis Python ;
— une **gestion d'état centralisée** : l'objet `AppState` maintient l'état de la session (messages, graphe, portrait, jalons, etc.), ce qui simplifie la gestion des nombreux événements ;
— une boucle d'événements (`ConversationFlow`) qui découple la **logique de conversation** de l'interface Streamlit.

4 Fonctionnement interne de l'agent conversationnel

4.1 Vue d'ensemble : de la phrase utilisateur au graphe mis à jour

À chaque message utilisateur, plusieurs **étapes** se déclenchent côté agent. L'idée générale est la suivante :

1. on stocke le message dans un *buffer* de messages récents ;
2. un **routeur** (petit LLM rapide) décide si on doit « recharger » les idées et le résumé global ou si on peut continuer avec le plan actuel ;
3. si besoin, on met à jour en parallèle :
   — un **document de résumé** de la conversation (ce que l'utilisateur veut, où il en est) ;

— un **document d'idées** (pistes de stratégies, prédictions possibles, dilemmes à proposer) ;

4. on appelle ensuite l'**agent principal**, le « Stratège maïeuticien », avec :
   — le résumé global ;
   — le document d'idées ;
   — le tampon de messages récents ;
   — une description textuelle du graphe actuel ;
   — le plan de conversation en cours.
5. l'agent principal renvoie :
   — une **nouvelle question ou réponse** courte à afficher ;
   — une mise à jour du **plan de conversation** (sa stratégie) ;
   — une **expression faciale** ;
   — éventuellement des **mutations de graphe** (nœuds/liaisons à créer ou modifier) ;
   — éventuellement un **jalon** atteint (par exemple « Bananes »).
6. l'interface applique les mutations, met à jour le graphe et l'affichage, et redonne la main à l'utilisateur.

4.2 Schéma des étapes de traitement

Nous avons représenté ce flux sous la forme d'un petit schéma que l'on peut compiler grâce à `tikz` :

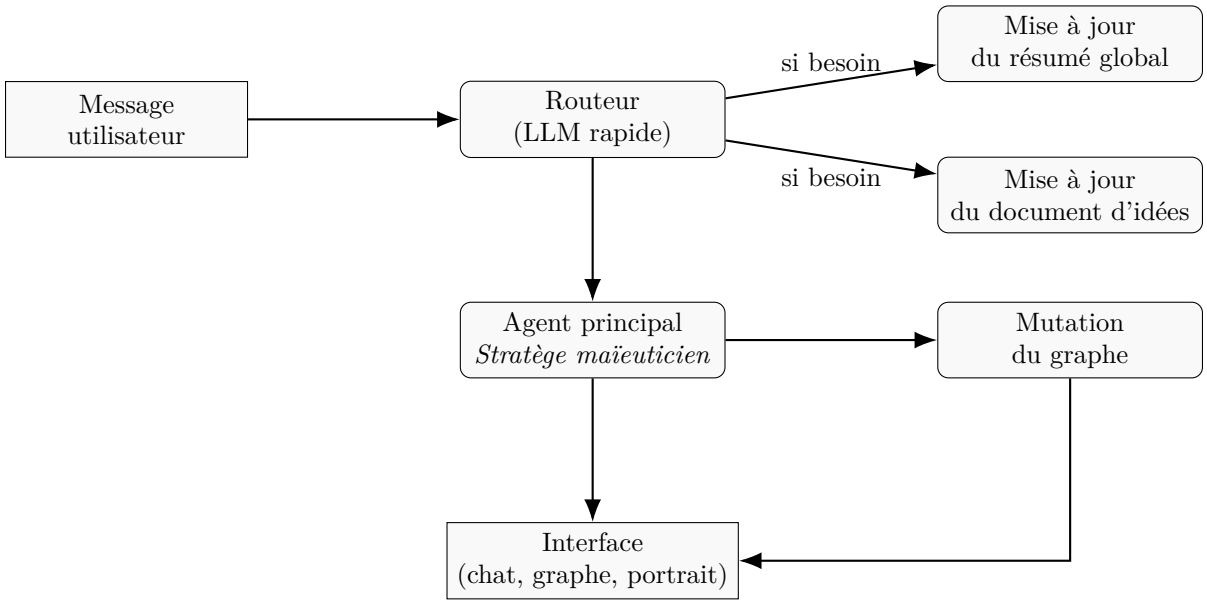

FIGURE 3 – Chaîne de traitement d'un tour de dialogue dans AYM.

Ce schéma illustre le rôle de chaque composant :
— le routeur régule le *rythme* de la conversation ;
— les « cerveaux » (résumé et document d'idées) maintiennent une mémoire de plus haut niveau ;
— l'agent principal fait le lien entre tout ça et la prochaine question à poser ;
— la couche interface (`Streamlit`) se charge de **rendre visibles** les changements : texte, graphe, portrait.

4.3 Document d'idées, stratégie de conversation et expressions faciales

Faire réfléchir le LLM « en coulisses ». Le *document d'idées* est une pièce centrale de l'architecture : c'est un texte que l'utilisateur ne voit jamais, mais qui sert de **boîte à idées** pour l'agent. Il permet au LLM de **réfléchir à l'avance** à partir du problème décrit par l'utilisateur, de générer une palette d'options et de conséquences possibles, puis d'utiliser cette matière pour poser des **questions plus ciblées** dans la suite de la conversation.

Concrètement, ce document est produit par un « cerveau » dédié (prompt `ideas_updater_system_prompt`) qui joue le rôle d'« Analyste de scénarios et de risques ». Il ne donne jamais de conseils prêts à l'emploi ; il fabrique plutôt des *dilemmes*, des *options* et des *conséquences* que le « Stratège maïeuticien » peut ensuite transformer en questions (« Est-ce que tu as envie d'essayer X ? Qu'est-ce que tu t'attendrais à voir si tu faisais ça ? »).

Quand (re)génère-t-on le document d'idées ? Nous ne régénérons pas ce document à chaque message, car ce serait coûteux en tokens et inutilement verbeux. C'est le **routeur** (`router_system_prompt`) qui décide, à chaque tour, si une mise à jour est nécessaire (`update_needed = True`) ou si l'on peut continuer avec les idées actuelles.

Il demande une mise à jour du document d'idées notamment lorsque :

- l'utilisateur introduit un **nouveau problème** ou un **nouvel enjeu clé** ;
- l'utilisateur semble **bloqué** (« Je ne sais pas quoi faire ») et manque de pistes concrètes ;
- le **graphe** ne contient pas assez de stratégies ou de prédictions claires pour relancer la discussion ;
- le document actuel n'est plus pertinent par rapport au sujet en cours (par exemple il parle encore de « solutions dialectiques » alors que l'utilisateur discute de pommes de terre) ;
- le document d'idées contient seulement une indication d'attente (« en attente du sujet de l'utilisateur ») alors que la personne est désormais engagée sur un vrai problème.

Dans tous ces cas, un modèle plus puissant (type GPT-5.1) est appelé pour régénérer ou enrichir le document d'idées en parallèle de la mise à jour du résumé global. Le reste du temps, on réutilise le même document sur plusieurs tours pour amortir le coût de génération.

Contenu typique du document d'idées. Le document d'idées suit une structure simple à deux étages, imposée par le prompt `ideas_updater_system_prompt` :

1. **Pistes de stratégies (options)** : une liste d'actions possibles, parfois techniques, parfois de contournement, parfois très pragmatiques (« faire une réparation de fortune », « appeler un pro », « tout laisser comme ça mais changer son organisation », etc.).
2. **Pistes de prédictions (conséquences & risques)** : pour chaque option, ce qui pourrait se passer en termes de coût, de délai, de confort, de risque social, d'impact émotionnel, etc.

L'idée est de fournir au Stratège une *carte mentale compressée* du problème, déjà structurée en relations *Stratégie* → *Conséquence*, afin qu'il puisse ensuite interroger l'utilisateur en se basant sur ces scénarios.

Ci-dessous, un **extrait ultra-condensé** d'un document d'idées pour une conversation à propos de la cuisson d'un steak :

> Substitution (soja / bouillon / parmesan).
> Demander du sel au voisin.
> Cuire sans sel, miser sur poivre / réaction de Maillard.
>
> Substitution → goût modifié, croûte fragilisée.
> Voisin → coût social, délai, risque hygiène.
> Sans sel → frustration, impression de repas fade.

Dans l'application, ce document reste **caché par défaut** à l'utilisateur : le Stratège s'en sert uniquement pour varier ses questions et proposer de nouvelles branches de graphe quand l'échange commence à tourner en rond.

L'utilisateur **peut le consulter** en cliquant sur un bouton explicite sous le portrait de l'agent, mais ce n'est **pas** une voie d'intéraction principale.

Stratégie de conversation (`conversation_plan`). En parallèle du document d'idées, l'agent principal maintient une **stratégie explicite de conversation**, renvoyée à chaque tour dans le champ `conversation_plan`. Ce plan joue un rôle de *méta-scénario* local :

- il résume en quelques points ce que l'agent compte faire dans les prochains tours (« 1. Clarifier l'objectif ; 2. Faire émerger deux options ; 3. Discuter des risques pour chacune ») ;
- il sert de garde-fou pour éviter que le LLM saute d'un sujet à l'autre sans terminer une branche de raisonnement ;
- il est **réinjecté dans le prompt d'instance** (`format_instance_prompt`) pour rappeler au modèle son propre plan au tour suivant, avec la consigne explicite « si ce plan aide encore, continue-le, sinon change-le ».

Cette stratégie est également affichée dans l'interface (panneau « Stratégie & Idées ») pour que l'utilisateur voie *où l'agent veut aller*. C'est donc un point de jonction entre le travail invisible (document d'idées, résumé, graphe) et la perception qu'a l'utilisateur de la **cohérence** de la discussion.

FIGURE 4 – Transparence du raisonnement : l'interface permet de déplier le panneau « Stratégie » pour voir le plan courant de l'agent ainsi que les idées de dilemmes générées en coulisses.

D'un point de vue coût, la stratégie est **courte** (quelques lignes) mais régénérée à chaque tour, tandis que le document d'idées est **plus long** mais mis à jour moins souvent. On joue donc sur ce différentiel : un document d'idées relativement volumineux, amorti sur plusieurs tours, et un plan léger qui peut s'adapter finement à chaque nouvelle réponse.

Pourquoi résumer la conversation ? Si l'on envoyait systématiquement *toute* l'historique brut de la conversation au LLM, le contexte serait rapidement saturé :

- les vieux messages de l'utilisateur occuperaient une grande partie du budget de tokens ;
- les messages les plus récents risqueraient d'être **dilués** au milieu d'un long historique, de la stratégie, du graphe et du document d'idées ;
- chaque appel deviendrait plus coûteux et plus lent.

Pour éviter cela, nous séparons la mémoire en trois couches :

1. un **tampon de messages récents** (`message_buffer`) qui ne contient que les derniers échanges, utilisés pour garder le ton et le contexte immédiat ;
2. un **résumé global** (`summary_doc`), mis à jour par le « Greffier du Projet » (`summary_updater_system_pro`) qui condense les objectifs, les stratégies déjà envisagées et l'état d'esprit de l'utilisateur ;

3. le **document d'idées** (`ideas_doc`), qui se concentre uniquement sur les options futures et leurs conséquences, sans recopier toute la conversation. En occupant une place importante dans le prompt, il aide le LLM à saisir le sujet de la conversation et comprendre que sa réponse doit porter sur ces perspectives.

De cette façon, le dernier message utilisateur reste **très visible** pour le modèle (dans le `message_buffer`), tandis que le résumé global et le document d'idées fournissent une mémoire de premier plan et une réserve de scénarios, sans que l'historique de messages brut prenne toute la place.

4.4 Contrôle de la narration : le système de Milestones.

Pour répondre à l'exigence de scénarisation malgré l'usage d'IA générative, nous avons implémenté un module de **Milestones** (Jalons). Ce module surveille en permanence l'état du graphe et de la conversation pour valider des étapes narratives précises (ex : "Le problème est défini", "Une stratégie risquée est identifiée", "Le graphe contient au moins 3 branches"). Cela permet de :

1. **Gamifier l'écriture** : L'utilisateur a des objectifs clairs à atteindre.
2. **Forcer la structure** : L'IA est incitée (via le prompt système qui lit ces jalons) à pousser la conversation vers les étapes non validées.

C'est ici que réside notre « scénario » : non pas dans les répliques, mais dans la grille d'objectifs obligatoires que le système impose subtilement à la conversation.
(Notez que ce système a été peu développé dans la version actuel, et que très peu de milestones sont disponibles. Ce n'est pas suffisant pour influencer le comportement de l'agent.)

4.5 Latence et organisation des appels LLM

Comme l'architecture repose sur plusieurs appels LLM par tour, la **latence** était un enjeu pratique :

— le routeur utilise un modèle **rapide** (GPT-4.1) pour être quasi instantané ;
— la mise à jour du résumé (*summary updater*) utilise aussi ce modèle rapide ;
— la mise à jour du document d'idées utilise un modèle **plus coûteux mais plus puissant** (GPT-5.1) pour générer des scénarios riches ;
— ces deux mises à jour se font *en parallèle* grâce à un `ThreadPoolExecutor` ;
— l'agent principal, lui aussi basé sur GPT-5.1, prend un peu plus de temps, mais c'est lui qui décide des mutations de graphe.

Pour que cette latence soit supportable du point de vue utilisateur :

— nous avons introduit des **WaitEvent** qui permettent d'afficher un message de type « AI is cooking » ou « Analyzing Context... » pendant que les appels s'exécutent ;
— la carte de l'agent passe en mode « réflexion » ou « dessin » grâce aux portraits correspondants ;
— la barre d'entrée se désactive clairement, ce qui évite de taper pendant que l'agent est en train de planifier sa réponse.

Cela transforme un temps mort technique en un temps « narratif » :

— on a l'impression que l'agent prend réellement le temps de réfléchir ;
— le passage en mode « dessin » pour la mise à jour du portrait vient renforcer ce ressenti.

5 Le graphe stratégique : structure et usage par le LLM

5.1 Rappel du modèle conceptuel

Le cœur du projet est le **graphe Stratégies & Prédictions**. Conceptuellement :

— un **nœud de stratégie** représente une action possible (par exemple « Parler à mon responsable », « Changer d'organisation », « Tester une solution temporaire ») ;
— un **nœud de prédiction** représente un résultat attendu, un état futur ou un objectif (« Moins de stress », « Projet terminé à temps », « Perdre en crédibilité »).

Les liens autorisés sont :

— **Stratégie** → **Prédiction** : ce qui se passe si on applique une stratégie ;
— **Prédiction** → **Stratégie** : comment on réagit si un certain état se produit ;
— **Prédiction** → **Prédiction** : relations entre états ou entre un état et un objectif plus global.

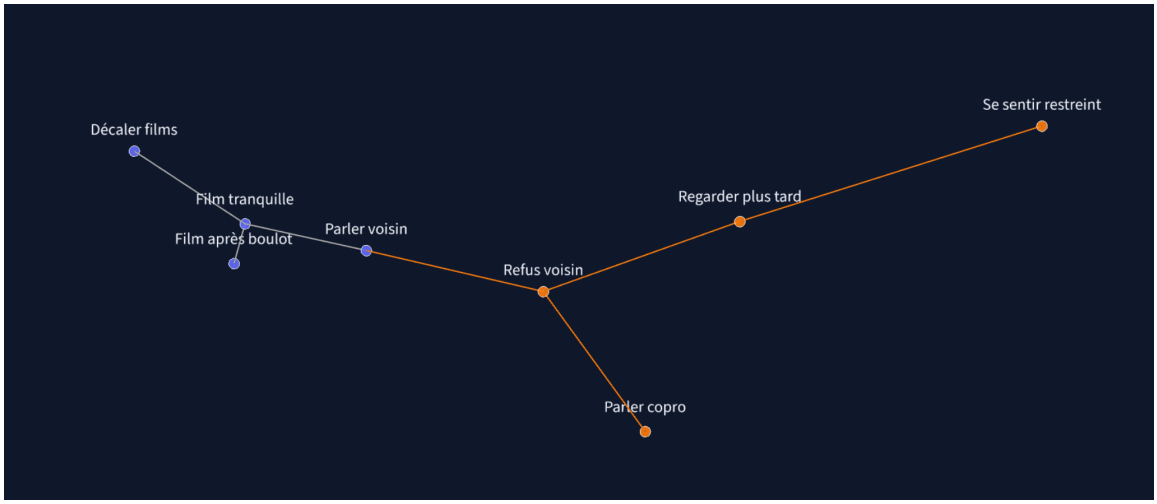

FIGURE 5 – Exemple de structure générée sur le thème « Voisin Bruyant ». Dans AYM, c'est une visualisation 3D interactive.

Le LLM est encouragé à garder des **labels de noeuds très courts** (1 à 3 mots) et des **contenus** qui débutent toujours par « [Stratégie] » ou « [Prédiction] », ce qui simplifie la lecture humaine et le traitement automatique.

5.2 Représentation interne et outil de mutation

En interne, le graphe est stocké dans une structure de type `GraphStorage` :

— une liste de nœuds (label, contenu textuel, indicateurs comme `new`) ;
— une liste d'arêtes (source, cible, indicateur `new`) ;
— le tout sérialisé dans un fichier JSON local.

Le LLM ne manipule jamais cette structure directement. À la place, il renvoie des instructions sous la forme d'un objet `GraphMutateInput`, qui contient :

— `nodes` : une liste de créations ou mises à jour de nœuds (avec option `connect_to` pour relier un nouveau nœud à des nœuds existants) ;
— `edges` : des arêtes explicites à créer entre nœuds déjà présents ;
— `delete_nodes` : des identifiants de nœuds à supprimer ;
— `delete_edges` : des paires (source, cible) à supprimer.

Une fonction asynchrone `mutate_graph` se charge ensuite d'appliquer ces mutations :

— elle marque d'abord tous les nœuds et arêtes existants comme « anciens » ;
— elle crée ou met à jour les nœuds demandés ;
— elle gère les *identifiants éphémères* (quand le LLM fait référence à des nœuds qui n'existent pas encore) ;
— elle ajoute les arêtes correspondantes en évitant les doublons ;
— elle supprime éventuellement certains nœuds et les arêtes incidentes.

Pour le rapport à l'utilisateur, un détail appréciable est la méthode `describe` de `GraphMutateInput`, qui produit un **résumé en français** des modifications (par exemple « a ajouté "Tourner la viande", a connecté "Tourner la viande" à "Devenir rouge" »). Ce résumé est affiché comme un message spécial dans le chat.

5.3 Ce que le LLM « voit » du graphe

Le LLM ne reçoit pas la structure brute du graphe, mais une *description textuelle* produite par une méthode de type `describe_for_llm` :

— nombre de nœuds et d'arêtes ;
— liste des labels et d'extraits de contenu ;
— représentation des arêtes sous la forme « (id :1, label :"Tourner") -> (id :2, label :"Devenir rouge") ».

Ce compromis permet de :

— donner au modèle une vision globale du graphe qu'il peut mentionner, sans exploser la longueur du contexte ;
— lui permettre de se servir du graph comme d'une forme de mémoire de la conversation complémentaire aux résumés des messages ;
— l'aider à choisir des identifiants cibles (pour `connect_to` ou `edges`) ;
— lui rappeler quelles stratégies et prédictions sont déjà présentes pour éviter les redondances.

5.4 Quand et pourquoi le graphe est modifié

Le graphe n'est pas mis à jour à chaque phrase automatiquement. L'agent est encouragé, via le prompt, à le faire lorsque :

— l'utilisateur formule clairement une **nouvelle stratégie** (« Je pourrais faire X ») ;
— l'utilisateur exprime une **nouvelle prédiction** ou un **nouvel objectif** (« Ce que je voudrais, c'est Y ») ;
— l'utilisateur clarifie une **relation causale** (« Si ça se passe comme ça, alors il arrivera Z »).

Cela évite de remplir le graphe avec des détails superficiels ou des reformulations. Le routeur, de son côté, peut déclencher une mise à jour du document d'idées s'il constate que :

— le graphe manque de stratégies concrètes ;
— on tourne un peu en rond sur un même nœud.

Du point de vue de l'utilisateur, le graphe devient ainsi une sorte de **synthèse des moments importants** de la conversation :

— les questions de l'agent « soulèvent » des nœuds ;
— quand une nouvelle option ou une nouvelle conséquence est stabilisée, elle est figée dans la structure ;
— l'utilisateur n'a pas besoin de gérer le détail de quand ça arrive : il le constate visuellement avec la mise à jour du graph et le petit message affiché dans le chat.

6 Aspects pratiques : historique, modèles, hébergement, multi-utilisateurs

6.1 Historique de conversation et mémoire

Côté interface, tous les messages sont stockés dans une liste `messages` au sein de `AppState` :

— chaque message a un rôle (« user », « ai », « graph », « summary », etc.) ;
— certains messages peuvent porter des *événements associés* (par exemple des annotations liées au graphe).

Côté agent, nous distinguons :

— un **tampon de messages récents** (`message_buffer`) qui sert de contexte immédiat pour le routeur et l'agent principal (afin de limiter la taille du prompt) ;
— un **document de résumé** (`summary_document`) qui condense l'état d'esprit et les objectifs de l'utilisateur à plus long terme ;
— un **document d'idées** (`ideas_document`) qui sert de réserve de scénarios et de dilemmes ;
— le **graphe** lui-même, qui est une mémoire structuraliste des stratégies et prédictions validées.

Cette séparation entre « mémoire courte » (buffer) et « mémoire longue » (résumé + graphe) permet :

— de ne pas saturer le contexte du LLM avec tout l'historique brut ;
— de favoriser un style de conversation concentré sur l'avenir et non sur la répétition du passé ;
— de conserver une trace stable (le graphe) qui survit au-delà de quelques tours de conversation.

6.2 Stratégie multi-modèles : GPT-5.1 et GPT-4.1

Pour équilibrer qualité de raisonnement et réactivité, nous avons choisi d'utiliser deux familles de modèles :

— **GPT-4.1** (ou équivalent) pour :
  — le routeur (décider s'il faut mettre à jour les documents internes) ;
  — la mise à jour rapide du résumé ;
  — certains retours succincts nécessitant peu de réflexion.
— **GPT-5.1** (ou équivalent) pour :
  — la génération du document d'idées (qui demande de produire beaucoup de scénarios et de conséquences) ;
  — l'agent principal, qui doit manipuler la structure de graphe tout en gardant un style de conversation précis.

En pratique :

— les appels à GPT-5.1 sont plus coûteux et plus lents, donc on les limite à ce qui a vraiment besoin de profondeur ;
— les appels à GPT-4.1 servent de « colle » : routage, résumés, décisions rapides. Ils ont aussi très peu de latence comparé aux 5.1.

6.3 Multi-utilisateurs avec Streamlit : contraintes et contournements

Streamlit n'est pas conçu à la base pour gérer des milliers d'utilisateurs simultanés avec des états complexes par session. Nous avons néanmoins mis en place quelques **mécanismes pour supporter le multi-utilisateur** raisonnable :

— tout l'état de la conversation est encapsulé dans un objet `AppState`, stocké dans `st.session_state["app_`
— chaque session navigateur a donc son propre `ConversationFlow`, son propre graphe en mémoire, ses propres messages ;
— les images et graphes sont mis en cache localement, ce qui réduit la charge de génération.

Cela reste :

— suffisant pour un contexte de démo ou de petit groupe ;
— mais évidemment pas dimensionné pour un service de production massif.

7 Conclusion, difficultés et pistes d'amélioration

7.1 Ce qui a bien marché

Sur le plan technique comme sur le plan expérimental, plusieurs points nous ont surpris positivement.

Compréhension des insertions dans le graphe. L'agent s'est révélé **très doué** pour décider *quand* ajouter un nœud au graphe et *comment* le relier :

— il comprend bien les formulations du type « Je pourrais faire X » ou « Je veux Y » ;
— il distingue assez proprement les stratégies (actions) des prédictions (résultats, objectifs) ;
— il recycle les nœuds existants au lieu de créer des doublons partout.

La structure `GraphMutateInput` et les instructions de prompt sur le format des nœuds (labels courts, contenu préfixé par [Stratégie]/[Prédiction]) ont clairement aidé à obtenir ce comportement.

Externalisation du plan de conversation. Le fait de faire expliciter par le LLM un **plan de conversation** (champ `conversation_plan`) a été très utile :

— pour nous, en tant que développeurs, cela donnait un feedback immédiat sur la manière dont le modèle comprenait les consignes ;
— pour l'utilisateur, cela offrait une fenêtre sur le « métaniveau » de la discussion, ce qui renforce la sensation de transparence.
— les idées en elles mêmes sont en général de bonne qualité.

7.2 Ce qui a été difficile

Tout n'a pas été simple, et certaines difficultés sont directement liées aux limites actuelles des LLM.

Faire générer des idées pour relancer l'utilisateur. Autant l'agent est bon pour **encoder** ce que l'utilisateur dit dans le graphe, autant il est moins naturel dans la génération d'**idées nouvelles** pour relancer la conversation :

— s'il n'est pas guidé, il a tendance à poser des questions très générales (« Et qu'en penses-tu ? ») qui n'apportent pas grand-chose ;
— ou au contraire des questions trop vagues ou trop précises, auxquelles l'utilisateur ne sait pas quoi répondre.

C'est précisément pour ça que nous avons introduit le **document d'idées** :

— ce document propose au LLM des pistes de stratégies (« Tenter une solution temporaire », « Attendre plus d'informations », « Demander de l'aide ») et des conséquences possibles ;
— l'agent se sert de ces pistes non pas pour « conseiller » directement, mais pour formuler des questions plus concrètes.

Nous avons constaté que :

— quand on demandait explicitement au modèle d'inclure **des suggestions dans ses questions** (« Est-ce que tu as pensé à X ? Que se passerait-il selon toi ? ») plutôt que de simplement rebondir vaguement, l'échange devenait immédiatement plus intéressant ;
— mais on a eu énormément de mal à faire **tenir cette consigne** tout au long de la conversation. Les LLMs semblent **rester bloqués** sur une idée initiale, malgré que la **quasi totalité de nos efforts de prompting** visaient à éviter ce phénomène.
— cette difficulté à même faire usage du document d'idées a été remarquées chez Gemini 3 pro et GPT 5.1 reasoning high, qui sont pourtant les **modèles state of the art** à l'heure de la rédaction de ce rapport. Ce problème est d'autant plus prononcé chez les modèles

plus anciens (GPT 4.1, Gemini 2.5 pro). Cela suggère que la **tâche en elle même est difficile**.
— On espère une amélioration avec l'arrivée des modèles futures, et de nouvelles stratégies de prompting.

Coût et fragilité du document d'idées. Le document d'idées, justement, est une structure puissante mais :
— **coûteuse** en nombre de tokens, donc en argent et en temps ;
— parfois **fragile** : si le LLM générateur sort trop de détails, le document devient difficile à exploiter ;
— difficile à garder **bien synchronisé** avec la conversation si on le met à jour trop souvent ou pas assez.

Le routeur nous a aidés à trouver un compromis :
— on ne met à jour le document que quand on voit un *vrai* changement de sujet ou un blocage ;
— le reste du temps, on s'appuie sur le plan de conversation et sur ce qui est déjà dans le graphe.

Éviter que le LLM saute d'un sujet à l'autre. Un autre problème classique est la **tendance du modèle à sauter de branche en branche** :
— il passe parfois trop vite à des sujets voisins sans avoir « fini » une idée ;
— parfois, le LLM étalait également son raisonnement dans sa réponse ;
— cela casse le naturel de la conversation.

Notre réponse a été la **stratégie explicite** :
— on demande au LLM de se doter d'un plan simple (par exemple « 1. Éclaircir l'objectif ; 2. Explorer deux options ; 3. Discuter des risques ») ;
— on lui rappelle ce plan à chaque tour dans le prompt d'instance, en lui laissant le choix de le continuer ou de le remplacer ;
— on expose ce plan à l'interface, ce qui responsabilise un peu le modèle (au sens où toute incohérence devient visible).

Cela ne règle pas tout, mais :
— on a beaucoup moins de « sauts » incompréhensibles ;
— le LLM sépare plus constamment raisonnement et réponse ;
— l'utilisateur peut mieux suivre le cheminement de la discussion.

7.3 Pistes d'amélioration

Plusieurs directions restent à explorer si l'on voulait pousser AYM plus loin.

Coupler l'agent à un système de récupération d'informations. Actuellement, l'agent est très centré sur *ce que dit l'utilisateur* et sur les idées produites par le LLM lui-même. On pourrait imaginer le coupler à un système de **recherche d'informations pertinentes** :
— sur des bases internes (par exemple les graphes d'autres utilisateurs, anonymisés) ;
— ou sur des ressources externes (documentation, articles).

L'idée serait surtout d'aider à la construction du document d'idées et surtout de proposer des exemples d'approches "de maieutique" adaptées à la situation de l'utilisateur.

Évaluer finement l'impact sur les utilisateurs. Dans le cadre de SI28, nous n'avons pas mené d'étude formelle d'usage. Il serait intéressant de :
— faire tester AYM à un panel plus large ;
— mesurer ce que les personnes retiennent de l'expérience ;

— voir si le graphe les aide réellement à prendre des décisions ou à clarifier des objectifs.

Cela pourrait passer par des questionnaires, des entretiens, ou une analyse qualitative des graphes produits.

Affiner la relation entre texte, graphes et images. Enfin, d'un point de vue plus artistique et multimédia (dans l'esprit de SI28), on pourrait travailler davantage sur :

— la mise en scène visuelle de l'évolution du graphe (animations, transitions, effets de mise en avant) ;
— la cohérence esthétique entre le style du portrait de l'agent et le style du graphe (couleurs, formes) ;
— la manière dont les expressions faciales pourraient directement refléter la structure du graphe (par exemple un visage plus tendu quand le graphe contient beaucoup de risques, plus serein quand les stratégies convergent).

7.4 Bilan

En résumé, AYM nous a permis de :

— expérimenter une **forme d'écriture interactive de soi**, où la conversation, le graphe et l'image d'un agent se répondent ;
— explorer concrètement comment **orchestrer plusieurs rôles LLM** (routeur, greffier, analyste d'idées, stratège) dans une même expérience, pour tenter de produire le comportement souhaité ;
— confronter ces choix à des contraintes très pragmatiques (latence, coût des appels, limite de contexte, multi-utilisateurs léger).

Le projet reste perfectible, mais il s'inscrit pleinement dans la logique de SI28 :

— utiliser le numérique non seulement comme un **support technique**, mais comme un **terrain d'expérimentation pour de nouvelles formes d'écriture** ;
— ici, l'écriture porte sur nos propres stratégies et prédictions, mises en scène par une IA qui ne cherche pas à nous remplacer, mais à nous faire penser.