

# Rapport Final - Yan Rabefaniraka

Yan Rabefaniraka, stagiaire de Meriem Ouederni et David Brunet

31 Juillet 2025

## Contents

<b>1</b>	<b>Sujet de stage</b>	<b>1</b>
<b>2</b>	<b>Intérêt de la mission en tant qu'étudiant avec une professeure pour tutrice</b>	<b>2</b>
<b>3</b>	<b>Ressenti - étudiant</b>	<b>2</b>
3.1	Utilisation de l'IA par le corps apprenant	2
3.1.1	Apprentissage	2
3.1.2	Est-ce que tu peux expliquer comment fonctionne la fonction bit2int (2025-07-31 22:25Z)	3
3.1.3	Facilitation des tâches pénibles	4
3.1.4	Triche	5
<b>4</b>	<b>Objectif: Co-cognition</b>	<b>5</b>
<b>5</b>	<b>Pistes d'amélioration vers la co-cognition</b>	<b>5</b>
<b>6</b>	<b>Tests menés sur des cas d'utilisation en code</b>	<b>6</b>
6.1	Jeu des allumettes - Cas grand public	6
6.1.1	Écriture d'un sujet	6
6.1.2	Résultats des tests	6
6.1.3	Conclusion - Jeu des allumettes	7
6.2	Ingénierie Dirigée par les Modèles - Filière Réseau de l'N7	7
6.2.1	Zones de test	7
6.2.2	Interpreting reachability analysis results in Petri nets (2025-07-17 09:25Z)	9
6.2.3	Detailed explanation of OCL elements in SimplePDL (2025-07-18 14:18Z)	10
6.2.4	Adding OCL constraints to SimplePDL model (2025-07-25 09:55Z)	12
6.2.5	Résultats et conclusion - IDM	14
6.2.6	Suite	14
<b>7</b>	<b>Préparation d'un notebook pour une possible formation</b>	<b>14</b>
7.1	Comparaison des modèles selon les cas d'utilisation	14
<b>8</b>	<b>Conclusions et pistes d'exploration</b>	<b>14</b>

### Remarques :

- Fichier engendré avec:

```
pandoc --toc -o ../pdfs/report1_yan_rabefaniraka.pdf report1_yan_rabefaniraka.md
```

- Par facilité d'écriture, on utilisera le masculin 'étudiant' pour qualifier les étudiant.es.

## 1 Sujet de stage

Pour ce stage, il m'est demandé en tant qu'étudiant en sciences du numérique d'établir une **étude de cas approfondie** sur l'utilisation actuelle par les étudiants de l'IA générative dans ce domaine— en particulier des

modèles de langage— et d’en déduire une possible utilisation plus efficace pour l’apprentissage. Mon expérience personnelle et mon statut d’étudiant sont un bon apport de départ, accompagnés par ma tutrice qui enseigne directement aux concernés.

Cette mission n’a **pas de filière cible** et n’est pas seulement dirigée vers celles spécialisées en informatique, tout d’abord parce que l’aide que l’IA pourrait fournir dans les études s’applique à tous les domaines, mais aussi parce que l’importance grandissante des petits cas d’usage de l’informatique dans les études supérieures (e.g savoir mettre sur un graphe avec *Python/Matlab* ses données d’expérience, ou encore faire des statistiques sur *RStudio*) permet de se questionner sur les facilitations que l’IA offre à celles et ceux dont le futur métier ne se trouve pas dans les sciences du numérique.

Ainsi, les tests effectués et les conclusions déduites de cette étude de cas— spécifiquement pour les sciences du numérique— doivent être applicables pour un **niveau débutant**, voire en particulier pour des langages haut niveau comme le *Python*.

Cependant, on peut aussi vérifier la puissance / les limites de l’utilisation de l’IA Générative dans l’apprentissage d’une matière complexe. Pour cela, **l’Ingénierie Dirigée par les Modèles**, enseignée (par Mme Ouederni aux 2ème années) à l’ENSEEIH est certainement choisie: C’est une matière suffisamment théorique mais avec des applications directes dans les différents processus de développement/industriels, avec une grande part de notions graphiques. Elle s’enseigne aussi avec 7 cours de *Travaux pratiques* sur machine, avec code et modélisation donc. On pourra ainsi vérifier ce que l’IA peut proposer à un étudiant qui cherche à l’utiliser pour travailler la matière en autonomie.

L’objectif final de cette étude de cas étant la création d’une formation / d’un module qui offrirait aux étudiants de tout horizon un **set d’outils** pour leur enseigner les bonnes pratiques, il n’est pas anodin de donner les bases de l’IA génératives à travers une courte vulgarisation du fonctionnement de la GenAI et des LLMs pour les non-initiés. Le public étant large il est aussi intéressant de mentionner les impacts environnementaux en rappelant les coûts en énergie et en eau de l’apprentissage et de la maintenance des services, de la place des utilisateurs dans la boucle (apprentissage actif, données personnelles) mais aussi des limites de l’IA *i.e* les hallucinations, les erreurs, les biais causés par le *data set*, les biais de confirmation de ChatGPT, etc. Mais, il est aussi essentiel de leur montrer à quel point ces outils sont puissants, en parlant de tout ce qui est inclus avec ChatGPT (compilateur LaTeX, Python), la gratuité de Github Copilot pour tout étudiant avec au choix l’utilisation de modèles de Claude Sonnet, GPT, et Gemini, ainsi que les modèles *offline*. Cela inclut donc un comparatif des différents modèles de langage gratuits actuellement sur le marché, par tâche et *cas d’utilisation* afin de comparer leur pertinence.

Enfin, cela implique donc d’établir des **critères de qualité/validation** de réponses selon le domaine, les prompts, l’IA, etc. Pour ce faire, plusieurs tests seront effectués et mis sur tableau tout au long du stage. Des templates seront également préparés pour d’éventuels *Jupyter Notebooks* (à lancer sur *Google Colab* par exemple). Ces derniers facilitent l’interactivité et augmenteront donc l’accessibilité du projet.

## 2 Intérêt de la mission en tant qu’étudiant avec une professeure pour tutrice

En tant qu’étudiant en sciences du numérique, je vais donc me servir de mes propres expériences avec l’IA Générative et vais partager l’usage que j’en fais ou que j’ai pu constater lors de ma 1<sup>ère</sup> année, que ce soit pour y apporter un regard critique ou m’en servir de base pour développer la suite du projet.

## 3 Ressenti - étudiant

Dans cette section, je vais donc étayer les différents usages de l’IA Générative que j’ai jusque là pu observer.

### 3.1 Utilisation de l’IA par le corps apprenant

#### 3.1.1 Apprentissage

On constate plusieurs cas d’utilisation courants que font les étudiants de l’IA générative dans l’apprentissage, en particulier de ChatGPT:

- **Trouver des solutions à des problèmes spontanés** qu'ils rencontrent lors de l'apprentissage d'un cours ou la résolution d'exercices, quand aucun professeur n'est à disposition pour y répondre. Cette utilisation est de plus en plus courante quand les étudiants résolvent des exercices pour lesquels il n'y a pas de correction claire et où ils ne sont donc pas assurés d'avoir les bonnes réponses, voire d'avoir compris le cours. Cela peut aussi s'appliquer à la programmation, quand la documentation pour une librairie est difficile à trouver ou à lire et qui incite régulièrement les étudiants à demander l'explication de lignes de code à un Chatbot. On peut par exemple penser aux différentes fonctions intégrées de *Matlab* dont la compréhension peut parfois être sinieuse. On peut aussi penser au développement *Web* où les étudiants peuvent demander à ChatGPT de leur expliquer les différentes balises *CSS* ou *HTML*.

→ Voici un exemple pour cette utilisation:

- Contexte:

```
%GENERATION DE L'INFORMATION BINAIRE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
bits = randi([0, 1], 1, N);
% bits_4aire = reshape(bits, [N/2, 2]);
dec_4aire = bit2int(bits', 2);
```

- Prompt: Est-ce que tu peux expliquer comment fonctionne la fonction matlab bit2int dans ce contexte ?
- Réponse:

### 3.1.2 Est-ce que tu peux expliquer comment fonctionne la fonction bit2int (2025-07-31 22:25Z)

*User*

Est-ce que tu peux expliquer comment fonctionne la fonction matlab bit2int dans ce contexte ?

*Assistant*

Je vais examiner le code pour comprendre le contexte et expliquer la fonction `bit2int`. Parfait ! Je peux maintenant expliquer le fonctionnement de la fonction `bit2int` dans ce contexte.

**3.1.2.1 Explication de la fonction bit2int dans ce contexte** La fonction `bit2int` en MATLAB convertit des séquences de bits en entiers décimaux. Dans votre code :

```
bits = randi([0, 1], 1, N);           % N = 1000 bits aléatoires (0 ou 1)
dec_4aire = bit2int(bits', 2);        % Conversion en entiers sur 2 bits
```

#### 3.1.2.1.1 Fonctionnement détaillé :

1. **Génération des bits** : `bits` est un vecteur ligne de 1000 bits aléatoires (0 ou 1)
2. **Transposition** : `bits'` transpose le vecteur pour en faire un vecteur colonne de  $1000 \times 1$
3. **Conversion par bit2int** :
  - **Syntaxe** : `bit2int(bits', 2)`
  - **Premier paramètre** : `bits'` - le vecteur colonne de bits
  - **Deuxième paramètre** : 2 - le nombre de bits par groupe à convertir
4. **Résultat** : `dec_4aire` contient 500 valeurs entières ( $1000 \div 2 = 500$ )

**3.1.2.1.2 Exemple concret** : Si `bits = [1, 0, 1, 1, 0, 0, 1, 0, ...]` (1000 bits), alors :

- `bits'` devient une colonne : `[1; 0; 1; 1; 0; 0; 1; 0; ...]`
- `bit2int(bits', 2)` groupe par 2 bits et convertit :
  - `[1; 0] → 2` ( $1 \times 2^1 + 0 \times 2^0$ )

- $[1; 1] \rightarrow 3 (1 \times 2^1 + 1 \times 2^0)$
- $[0; 0] \rightarrow 0 (0 \times 2^1 + 0 \times 2^0)$
- $[1; 0] \rightarrow 2 (1 \times 2^1 + 0 \times 2^0)$
- etc.

**3.1.2.1.3 Contexte télécommunications :** Cette conversion est nécessaire car :

- **M = 4** : modulation 4-aire (4-ASK ou 4-QAM)
- $\log_2(4) = 2$  : chaque symbole porte 2 bits d'information
- Les fonctions de mapping (`pammod`, `qammod`) attendent des entiers de 0 à 3, pas des bits

Donc `dec_4aire` contient des valeurs dans  $\{0, 1, 2, 3\}$  qui seront ensuite mappées sur les symboles de constellation par les fonctions de modulation.

- 
- **Expliquer/résumer un cours** qu'ils n'ont pas compris ou dont ils veulent vérifier leur propre compréhension. Cela peut s'apparenter à de l'aide personnalisée, avec les étudiants qui fournissent la section du cours qu'ils n'ont pas comprise. De même, certains s'en servent pour générer des questions typiques ou des quiz, comme des équivalents à la plateforme *Quizlet*.

### 3.1.3 Facilitation des tâches pénibles

Une utilisation courante est aussi celle de la **facilitation des tâches fastidieuses** (répétitives, peu challengeantes) en passant par l'IA. On rencontre le plus ce cas d'utilisation lors de de TPs ou de projets qui nécessitent du code répétitif et peu demandant en réflexion technique pour les étudiants. On peut notamment penser aux méthodes répétitives en programmation orientée objet, comme des *getters* ou des *setters*, ou encore aux matières dont le code n'est pas le sujet principal mais un support d'application (algèbre linéaire, automatique ou télécommunications par exemple).

→ Voici un exemple pour cette utilisation:

- Prompt: Génère-moi en java les getters et setters pour une classe appelée Cercle ayant pour attributs un rayon (flottant), une coordonnée x (flottant) et une coordonnée y. Prévois aussi la gestion d'exception.
- Réponse(réponse tronquée):

```
// Getter pour le rayon
public float getRayon() {
    return rayon;
}

// Setter pour le rayon avec validation
public void setRayon(float rayon) throws IllegalArgumentException {
    if (rayon <= 0) {
        throw new IllegalArgumentException("Le rayon doit être strictement positif");
    }
    this.rayon = rayon;
}

// Getter pour la coordonnée x
public float getX() {
    return x;
}

// Setter pour la coordonnée x
public void setX(float x) {
    this.x = x;
}
```

```
// Getter pour la coordonnée y
public float getY() {
    return y;
}

// Setter pour la coordonnée y
public void setY(float y) {
    this.y = y;
}
```

(Pour information, utiliser `IllegalArgumentException` est une bonne pratique mais n'est pas forcément approprié dans notre cas. Il aurait suffi d'utiliser des *asserts*.)

### 3.1.4 Triche

On observe parfois des cas où les étudiants font faire tout leur projet / TP par l'IA, sans être capable de reproduire ni *a minima* d'expliquer le résultat. Cela a un apport nul pour l'apprentissage et est donc le cas qu'on cherche à éviter. On peut se baser notamment sur la récente étude menée par des chercheurs du MIT qui ont conclu à une [baisse des capacités cognitives liées à l'utilisation exclusive de ChatGPT par des étudiants pour écrire leurs dissertations](#); En effet, il ne faudrait pas que la facilité d'utilisation de l'IA nuise à l'intégrité réflexive des apprenants, en particulier quand ces derniers n'ont pas encore acquis l'expertise dans le domaine concerné.

## 4 Objectif: Co-cognition

L'objectif visé serait pour l'étudiant d'atteindre la co-cognition: Un stade où, ni l'IA ni l'étudiant n'auraient été capables de réaliser le code sans l'autre. C'est-à-dire que le code final serait bien meilleur que celui proposé en un *prompt* par l'IA, et que l'étudiant se voit capable d'expliquer chaque étape puisqu'il aura directement contribué à chacune. Cela implique que ce dernier sait exactement ce qu'il veut et qu'il est capable d'évaluer le code que lui propose l'IA, de revenir dessus, éventuellement de l'améliorer ou de le corriger.

## 5 Pistes d'amélioration vers la co-cognition

Les principes que j'ai utilisés pour essayer d'approcher la co-cognition (/ co-construction) sont:

- **Role prompting** → Définir un rôle et des compétences spécifiques à l'IA (e.g: "Tu es un programmeur fonctionnel"). Cela permet d'ajouter du contexte qui dirige la génération de l'IA vers un résultat plus spécialisé; C'est en particulier efficace avec des IA généralistes comme **ChatGPT**. Cela s'est révélé moins impactant avec **Github Copilot** qui est *fine-tuned* avec les répertoires publics **Github**, et qui peut donc très bien faire le travail sans.
- **Contexte détaillé** → Fournir les spécifications complètes du projet c'est-à-dire les consignes du projet données par le professeur concernant le code et le raisonnement, mais aussi ce qu'on attend exactement de l'IA, par exemple "expliquer le code/la syntaxe" ou encore "écrire un squelette modulaire du code".
- **Collaboration par itération**: Être capable d'interagir avec les réponses de l'IA. Cela implique de travailler de son côté sur ce qu'elle renvoie, comme corriger son code, repérer les erreurs de compréhension (ou hallucinations), les manquements, ou simplement améliorer les prémisses qu'elle propose. La co-cognition/co-construction ressemblera inévitablement à une discussion entre deux collègues qui améliorent ou questionnent tour à tour leur proposition.
- **Connaissances établies dans le domaine**: Pour que tous les points précédents soient efficaces, cela nécessite de l'utilisateur qu'il soit au moins légèrement éduqué voire expert dans le domaine sur lequel il utilise l'IA. En clair, il a besoin de connaître les bases de la syntaxe du langage de code/modélisation utilisé, les détails exacts du sujet avec les attentes et contraintes, et est capable de questionner les générations de l'IA. Son but est de rebondir sur ces dernières pour produire encore mieux; elles peuvent servir de base de réflexion ou de base de code pour quelque chose qu'il saurait faire correctement/décemment tout seul.

## 6 Tests menés sur des cas d'utilisation en code

Pour l'étude de cas approfondie, nous avons pris deux sujets distincts dans leur exigence. Cela nous permettra de nous faire une idée sur la viabilité d'utiliser une IA pour un cas accessible comme pour un cas plus complexe.

### 6.1 Jeu des allumettes - Cas grand public

Le jeu des allumettes est un sujet de projet très court que j'ai eu à réaliser en **Ada** puis en **Java**. La légère complexité vient des contraintes imposées qui sont claires et rigides, et qui permettent d'évaluer la capacité d'un étudiant à implémenter un cahier des charges très explicite dans ses attentes.

Pour plus de détails, voir [ici](#).

#### 6.1.1 Écriture d'un sujet

La première étape a été de réécrire simplement le [sujet](#) du jeu des allumettes. Il inclut les règles du jeu et conditions de victoire, ainsi que les contraintes de programmation et exigences. J'y aussi mis des extraits d'affichage suffisamment complet pour qu'il soit reproductible, comme ici:

```
Il reste 13 allumettes sur le tas.
o o o o o   o o o o o   o o o
| | | | |   | | | | |   | | |
| | | | |   | | | | |   | | |
| | | | |   | | | | |   | | |

Combien voulez-vous en prendre ?
> 13
Impossible ! Prise invalide. (13 > 3).

Combien voulez-vous en prendre ?
> 0
Impossible ! Prise invalide. (0 < 1).

Combien voulez-vous en prendre ?
> 1
Vous retirez 1 allumette au tas.

Il reste 12 allumettes sur le tas.
o o o o o   o o o o o   o o
| | | | |   | | | | |   | |
| | | | |   | | | | |   | |
| | | | |   | | | | |   | |

A mon tour.
> 3
```

Cela pourra servir de base pour évaluer les réponses de l'IA et ainsi savoir comment mieux poser les prompts à partir du sujet directement. Quant à la qualité des prompts, elle sera aussi auto-évaluée selon la grille du [PPAi6](#), au même titre que la qualité des réponses sera évaluée selon le code obtenu.

#### 6.1.2 Résultats des tests

- **Consommation passive:**
  - Constat → L'affichage des allumettes n'est pas exactement comme demandé; L'IA n'a pas correctement généré la stratégie **distrain**, et a surtout inversé le gagnant/perdant. De plus, le code en lui-même montre des lacunes avec beaucoup de structures de contrôle imbriquées et une mauvaise exploitation de la polyvalence de Python. On remarque clairement que cette manière de se servir de l'IA est terriblement inefficace; cette dernière fait des erreurs de compréhension et doit compléter des pans importants de la consigne qui ont été omis dans le contexte. Pour autant, on peut voir la puissance de l'IA pour générer du code qui fonctionne, même si ce dernier ne remplit pas les contraintes du sujet. On verra aussi plus tard

que certaines erreurs que fait ChatGPT (Copilot moins) se poursuivent dans les tests suivants malgré un meilleur prompting.

- **Création de contenu:**
  - Constat → Les deux IA utilisées ont généré une structure de départ plutôt bonne et facilement complétable/compréhensible par l’humain derrière, similaire à un raffinage. Pour autant, Copilot avec Claude Sonnet continue de sur-générer le code et en fait plus que nécessaire. Cela s’avèrerait utile pour la co-crédation nonobstant.
- **Tentative de co-crédation**
  - Constat → Le constat fait est que la co-crédation est possible avec les LLMs sur ce genre de sujets simples, avec un code produit de qualité, respectant les contraintes et attentes du sujet. Cependant, cela demande à l’utilisateur de savoir interagir intelligemment avec l’IA: Il doit savoir prompter efficacement avec du role prompting, fournir un contexte détaillé avec les attentes et contraintes exactes du cahier des charges, et être capable de corriger les erreurs de l’IA, ou de rebondir sur ses propositions pour avancer de son côté. Cela implique donc que l’utilisateur ait au moins les bases du langage de programmation utilisé, et surtout une compréhension précise du sujet. Il doit pouvoir communiquer efficacement ce qu’il attend de l’IA, et doit pouvoir interagir avec l’IA comme un assistant à qui il montre ses avancées et demande une suggestion ou une correction. Dans le cas du jeu des allumettes, l’IA– en particulier ChatGPT– a beau avoir produit un code respectable, les biais d’apprentissage produisent des erreurs de compréhension de consigne, et les lacunes en bonnes pratiques de programmation rendent la correction obligatoire. Aussi, il semblerait aussi que Github Copilot (ou Claude Sonnet par extension) soit plus efficace pour la co-crédation que ChatGPT si on s’inspire des résultats précédents, malgré sa tendance à en faire trop et à donc laisser moins de marge de réflexion personnelle à l’utilisateur.

### 6.1.3 Conclusion - Jeu des allumettes

Ce qu’on peut conclure de cette étude de cas est que dans le contexte de l’apprentissage de la programmation, les LLMs sont capables de **produire du code de qualité**, mais nécessitent activement que **l’utilisateur interagisse avec pour corriger ou rebondir sur leurs propositions**. Pour que l’apprentissage et la co-crédation soient efficaces, il faut donc que l’utilisateur ait une compréhension précise du sujet et une certaine expertise dans le langage utilisé. Pour autant, il semblerait que l’IA puisse parfois **faire beaucoup trop et mêche le travail de l’utilisateur**, ce qui peut être *contre-productif* si ce dernier tente de s’exercer ou de se former. Et à cela s’ajoute aussi les IA qui font des **erreurs** ou qui **hallucinent** dans leur réponse, comme on a pu observer ChatGPT le faire dans tous les tests sans exception. En clair, le mieux serait pour les étudiants de d’abord commencer leur raffinage et d’ensuite s’aider de l’IA pour convertir leur travail. Ils pourraient ensuite travailler en tandem pour améliorer le produit et corriger les éventuels problèmes.

## 6.2 Ingénierie Dirigée par les Modèles - Filière Réseau de l’N7

L’*Ingénierie Dirigée par les Modèles* est une matière relativement complexe enseignée à l’ENSEEIH. L’itération choisie est celle pour la filière **Réseau** de l’école, qui n’est donc pas forcément composée d’étudiants experts en informatique. Ainsi, on peut observer l’utilité de l’IA dans l’apprentissage et l’application de langages de modélisation, pour des étudiants dont le domaine d’expertise visé n’est pas la programmation ou le logiciel. Cette section part directement du postulat qu’on veut la co-crédation et utilise donc les points mentionnés dans la section 5.

Pour plus de détails, voir [ici](#).

### 6.2.1 Zones de test

En tant qu’élève peu expérimenté en IDM et dont mes connaissances se limitent à de l’apprentissage autonome, j’ai utilisé l’IA pour:

- **Expliquer les graphes de marquage :**
  - Fichier *Tina* à expliquer:

```
Tina version 2.8.0 -- 03/14/06 -- LAAS/CNRS
```

```
mode -R
```

```
INPUT NET -----
```

```

parsed net noname

## Description textuelle du reseau (voir FAQ)

5 places, 4 transitions

net noname
tr OqpA A idle -> WaitB
tr OqpAB B WaitB -> A B idle
tr OqpB B idle -> WaitA
tr OqpBA A WaitA -> A B idle
pl A (1)
pl B (1)
pl idle (2)

0.000s

REACHABILITY ANALYSIS -----

bounded
## Reseau Borne

4 marking(s), 6 transition(s)
## Le graphe des marquages est constitué
## de 4 états (marquages) et 6 transitions

MARKINGS:

## Liste des marquages

0 : A B idle*2
1 : B WaitB idle
2 : WaitA WaitB
3 : A WaitA idle

## Liste des transitions

REACHABILITY GRAPH:

0 -> OqpA/1, OqpB/3
1 -> OqpAB/0, OqpB/2
2 ->
3 -> OqpA/2, OqpBA/0

0.000s

LIVENESS ANALYSIS -----

not live
## reseau non vivant

1 dead marking(s), 1 live marking(s)
0 dead transition(s), 0 live transition(s)

```



```

dead marking(s): 2
## un marquage bloquant

## Analyse des composantes fortement connexes
## A voir si on a du temps.

STRONG CONNECTED COMPONENTS:

1 : 0 1 3
0 : 2

SCC GRAPH:

1 -> 0qpA/1, 0qpB/1, 0qpAB/1, 0qpB/0, 0qpA/0, 0qpBA/1
0 ->

0.010s

ANALYSIS COMPLETED -----

```

– Explication de l'IA:

---

## 6.2.2 Interpreting reachability analysis results in Petri nets (2025-07-17 09:25Z)

### *User*

Tu es expert en ingénierie dirigée par les modèles et tu as pour rôle d'interpréter et comprendre le résultat du graphe de marquage réalisé par le logiciel Tina, que tu as en contexte. Le fichier d'origine simule une situation d'interblocage, et les résultats sont ceux de l'atteignabilité. Pourrais-tu expliquer en détail ce que dit ce fichier ?

### *Assistant*

D'après le fichier de résultats Tina, voici l'analyse détaillée de ce réseau de Petri qui modélise une situation d'interblocage :

#### 6.2.2.1 Structure du réseau Le réseau contient :

- \* **5 places** : A, B, idle (avec 2 jetons initialement), WaitA, WaitB
- \* **4 transitions** : 0qpA, 0qpAB, 0qpB, 0qpBA

Les transitions représentent :

- \* 0qpA : le processus A demande une ressource
- \* 0qpB : le processus B demande une ressource
- \* 0qpAB : libération des ressources par A et B
- \* 0qpBA : libération des ressources par B et A

#### 6.2.2.2 Analyse d'atteignabilité Le graphe des marquages révèle 4 états possibles :

- \* **État 0** (initial) : A B idle\*2 - Les deux processus sont actifs avec 2 ressources libres
- \* **État 1** : B WaitB idle - Le processus A attend, B est actif, 1 ressource libre
- \* **État 2** : WaitA WaitB - **ÉTAT D'INTERBLOCAGE** - Les deux processus attendent mutuellement
- \* **État 3** : A WaitA idle - Le processus B attend, A est actif, 1 ressource libre

#### 6.2.2.3 Graphe de transitions

État 0 → États 1 ou 3 (via 0qpA ou 0qpB)  
 État 1 → États 0 ou 2 (via 0qpAB ou 0qpB)  
 État 2 → AUCUNE transition possible (interblocage)  
 État 3 → États 2 ou 0 (via 0qpA ou 0qpBA)

**6.2.2.4 Analyse de vivacité** Le réseau est **non vivant** car :

- \* Il existe **1 marquage mort** (état 2) où aucune transition n'est possible
- \* Cet état représente l'interblocage classique où chaque processus détient une ressource et attend celle détenue par l'autre

**6.2.2.5 Composantes fortement connexes**

- \* **SCC 1** : États {0, 1, 3} - forme un cycle où les processus peuvent évoluer
- \* **SCC 0** : État {2} - état isolé sans sortie (puits)

Ce modèle illustre parfaitement le problème d'interblocage mutuel entre deux processus concurrents partageant des ressources limitées.

- 
- Constat → Elle a servi à m'expliquer les graphes de marquage du cas d'interblocage sur Tina. Elle a su très bien expliquer tous les éléments pour quelqu'un comme moi qui s'y connaît peu: Elle détaille les états, comprend correctement les transitions et explique pertinemment d'où vient l'interblocage. Cette utilisation pourrait être utile pour comprendre un concept du cours et la syntaxe de modélisation associée. Cependant, l'IA a clairement mâché le travail pour moi, et il m'est difficile de dire si j'aurais pu comprendre le graphe sans son aide. De même, le manque de réflexion de mon côté interroge sur la permanence des informations tout juste acquises dans ma mémoire (sans l'effort personnel, compliqué de mémoriser). Il est donc difficile de dire si l'IA est utile pour apprendre ou si elle est contre-productive dans ce cas.

- **Apprendre la syntaxe OCL et comprendre le métamodèle de SimplePDL en OCL:**

- Explication de l'IA:
- 

## **6.2.3 Detailed explanation of OCL elements in SimplePDL (2025-07-18 14:18Z)**

*User*

Tu es expert en OCL. Peux-tu m'expliquer en détail les éléments apparaissant dans le fichier en contexte ? N'hésite pas à détailler la syntaxe.

*Assistant*

Je vais vous expliquer en détail ce fichier OCL (Object Constraint Language) qui définit des contraintes pour un modèle SimplePDL.

### **6.2.3.1 Structure générale du fichier**

```
import 'SimplePDL.ecore'  
package simplepdl
```

#### **6.2.3.1.1 Import et package**

- \* **import** : Importe le métamodèle Ecore correspondant
- \* **package** : Définit l'espace de noms pour les contraintes

### **6.2.3.2 Contraintes sur la classe Process**

```
context Process  
inv warningSeverity: false  
inv withMessage('Explicit message in process ' + self.name + ' (withMessage)': false  
inv errorSeverity: null
```

#### **6.2.3.2.1 Contraintes de test et de configuration**

- \* **context Process** : Spécifie que les contraintes s'appliquent à la classe Process
- \* **inv** : Définit une invariante (contrainte qui doit toujours être vraie)
- \* Ces contraintes semblent être des exemples de test :
  - **warningSeverity** : Génère un avertissement (retourne **false**)
  - **withMessage** : Montre comment inclure un message personnalisé avec concaténation de chaînes
  - **errorSeverity** : Génère une erreur (retourne **null**)

```
context Process
inv validName('Invalid name: ' + self.name):
    self.name.matches('[A-Za-z_][A-Za-z0-9_]*')
```

#### 6.2.3.2.2 Contrainte de validation du nom

- \* **self** : Référence à l'instance courante de Process
- \* **matches()** : Méthode OCL pour vérifier qu'une chaîne correspond à une expression régulière
- \* **Expression régulière** : [A-Za-z\_][A-Za-z0-9\_]\*
  - Premier caractère : lettre ou underscore
  - Caractères suivants : lettres, chiffres ou underscores
- \* **Message d'erreur** : Inclut le nom invalide dans le message

```
context ProcessElement
def: process(): Process =
    Process.allInstances()
        ->select(p | p.processElements->includes(self))
        ->asSequence()->first()
```

#### 6.2.3.3 Définition d'opération sur ProcessElement

- \* **def:** : Définit une nouvelle opération
- \* **process(): Process** : Opération qui retourne le Process contenant cet élément
- \* **Process.allInstances()** : Récupère toutes les instances de Process
- \* **->select()** : Filtre les éléments selon une condition
- \* **p | p.processElements->includes(self)** : Expression lambda - garde les Process qui contiennent cet élément
- \* **->asSequence()->first()** : Convertit en séquence et prend le premier élément

```
context WorkSequence
inv successorAndPredecessorInSameProcess('Activities not in the same process : '
    + self.predecessor.name + ' in ' + self.predecessor.process().name+ ' and '
    + self.successor.name + ' in ' + self.successor.process().name
):
    self.process() = self.successor.process()
    and self.process() = self.predecessor.process()
```

#### 6.2.3.4 Contrainte sur WorkSequence

- \* **Contrainte métier** : Vérifie que le prédécesseur et le successeur d'une séquence de travail sont dans le même processus
- \* **Message d'erreur détaillé** : Indique les noms des activités et leurs processus respectifs
- \* **Condition** : Utilise l'opérateur **and** pour vérifier deux égalités
- \* **Navigation** : Utilise la méthode **process()** définie précédemment

#### 6.2.3.5 Syntaxe OCL utilisée

- \* **Opérateurs de collection** : ->select(), ->includes(), ->asSequence(), ->first()
- \* **Opérateurs logiques** : and
- \* **Concaténation de chaînes** : +
- \* **Navigation dans le modèle** : .name, .predecessor, .successor
- \* **Expressions lambda** : p | condition
- \* **Messages personnalisés** : Tous les invariants incluent des messages d'erreur explicites

Ce fichier illustre bien les capacités d'OCL pour définir des contraintes métier complexes avec des messages d'erreur informatifs.

- Constat → J'ai utilisé l'IA pour m'aider à comprendre la syntaxe OCL et le métamodèle de SimplePDL. Elle a pu m'expliquer les concepts de manière claire et concise, et la réponse est clairement réutilisable pour réviser et renforcer ma compréhension de la syntaxe OCL. Cependant, la difficulté de l'IA à comprendre certains éléments du métamodèle montre qu'elle n'est pas infaillible et qu'il est donc nécessaire de

comprendre le sujet modélisé. Ainsi, c'est définitivement une bonne manière de réviser la syntaxe OCL, mais il est impossible d'apprendre avec l'IA seule et sans avoir au préalable une compréhension du sujet.

- **Écrire des contraintes SimplePDL en OCL:**

- Réponse de l'IA:

#### 6.2.4 Adding OCL constraints to SimplePDL model (2025-07-25 09:55Z)

##### *User*

Tu es un expert en OCL (Object constraint language). Tu m'aides à écrire de nouvelles contraintes pour le SimplePDL à l'aide de ceci:

```
import 'SimplePDL.ecore'

package simplepdl

context Process
inv warningSeverity: false
inv withMessage('Explicit message in process ' + self.name + ' (withMessage)': false
inv errorSeverity: null

context Process
inv validName('Invalid name: ' + self.name):
    self.name.matches('[A-Za-z_][A-Za-z0-9_]*')

context ProcessElement
def: process(): Process =
    Process.allInstances()
        ->select(p | p.processElements->includes(self))
        ->asSequence()->first()

context WorkSequence
inv successorAndPredecessorInSameProcess('Activities not in the same process : '
    + self.predecessor.name + ' in ' + self.predecessor.process().name+ ' and '
    + self.successor.name + ' in ' + self.successor.process().name
):
    self.process() = self.successor.process()
    and self.process() = self.predecessor.process()

endpackage
```

je veux ajouter 4 nouvelles contraintes: trace-moi le chemin pour y arriver.

##### *Assistant*

Je vais vous guider pour ajouter 4 nouvelles contraintes OCL à votre fichier SimplePDL. Voici le chemin étape par étape :

**6.2.4.1 Étape 1 : Analyser la structure existante** D'abord, examinez la structure de votre fichier SimplePDL.ecore pour identifier les classes et attributs disponibles. Les contraintes existantes montrent qu'il y a au moins : - Process avec un attribut name - ProcessElement - WorkSequence avec predecessor et successor

**6.2.4.2 Étape 2 : Choisir les types de contraintes à ajouter** Voici 4 types de contraintes courantes pour SimplePDL :

1. **Contrainte sur les noms uniques des activités**
2. **Contrainte sur les dépendances cycliques**
3. **Contrainte sur les ressources**
4. **Contrainte sur la cohérence des types de liens**

```

import 'SimplePDL.ecore'

package simplepdl

context Process
inv uniqueName('')

// ...existing code...

-- Contrainte 1 : Noms uniques des activités dans un processus
context Process
inv uniqueActivityNames('Duplicate activity names in process ' + self.name):
    self.processElements->select(oclIsTypeOf(WorkDefinition))
        ->collect(oclAsType(WorkDefinition).name)->isUnique(n | n)

-- Contrainte 2 : Pas de dépendances cycliques
context WorkSequence
inv noCyclicDependencies('Cyclic dependency detected'):
    not self.predecessor.allSuccessors()->includes(self.predecessor)

-- Contrainte 3 : Les ressources utilisées doivent être déclarées
context WorkDefinition
inv declaredResources('Undeclared resource used'):
    self.usedResources->forall(r |
        self.process().processElements->select(oclIsTypeOf(Resource))
            ->exists(res | res.oclAsType(Resource) = r))

-- Contrainte 4 : Cohérence des types de liens de séquence
context WorkSequence
inv validLinkType('Invalid link type for sequence'):
    self.linkType = WorkSequenceType::startToStart implies
        (self.predecessor.oclIsTypeOf(WorkDefinition) and
         self.successor.oclIsTypeOf(WorkDefinition))

endpackage

```

#### 6.2.4.3 Étape 3 : Implémenter les contraintes

**6.2.4.4 Étape 4 : Définir les opérations auxiliaires si nécessaire** Si certaines contraintes nécessitent des opérations auxiliaires, ajoutez-les :

```

-- Opération auxiliaire pour détecter les cycles
context WorkDefinition
def: allSuccessors(): Set(WorkDefinition) =
    self.outgoingSequences->collect(successor)->asSet()
        ->union(self.outgoingSequences->collect(successor)
            ->collect(allSuccessors())->flatten()->asSet())

```

#### 6.2.4.5 Étapes suivantes :

1. **Vérifiez** que les noms de classes et attributs correspondent à votre modèle Ecore
2. **Testez** chaque contrainte individuellement
3. **Adaptez** les messages d'erreur selon vos besoins
4. **Validez** avec des instances de test

Souhaitez-vous que je détaille l'une de ces contraintes ou que je propose d'autres types de contraintes spécifiques à votre domaine ?

- Constat → J’ai demandé à l’IA de m’aider à ajouter certaines contraintes au métamodèle de SimplePDL. En donnant le fichier pour contexte, il est capable de proposer des contraintes cohérentes avec le métamodèle. L’IA comprend bien la logique des dépendances entre activités et peut suggérer des invariants pertinents, bien qu’il faille parfois corriger la syntaxe OCL spécifique ou la signification de certains contextes. Cela montre que l’IA peut être utile pour écrire du code OCL, mais qu’elle nécessite une compréhension préalable du métamodèle et de la syntaxe OCL pour être efficace. En somme la conclusion est la même: l’IA semble être un bon outil pour aider à écrire des métamodèles mais nécessite d’être capable de comprendre intégralement le sujet sans quoi elle fait le gros du travail, incluant des erreurs sur le chemin.

### 6.2.5 Résultats et conclusion - IDM

À ce stade de l’étude de cas, on peut dire dans l’ensemble que l’IA a actuellement les capacités pour aider à comprendre les principes et les différentes syntaxes liés à la modélisation de métamodèles, mais qu’elle a encore des difficultés évidentes à certains éléments du sujet et écrit des bêtises. Cela est symptomatique du domaine de l’IDM qui est un peu plus niché et complexe, et qui est donc moins bien compris par les LLMs. À noter que l’IA mâche clairement le travail pour certaines tâches, et qu’il est donc nécessaire de connaître le sujet au préalable pour bénéficier correctement de son aide.

### 6.2.6 Suite

Pour voir la pertinence de l’utilisation de l’IA en IDM, il reste trois tests à essayer de réaliser avant la fin du stage:

- TP sur XTest
- TP sur Sirius
- TP sur Acceleo

Ces tests donneront sûrement une idée plus précise de la place que pourrait prendre l’IA dans l’apprentissage de sujets aussi complexes que l’IDM, et de la pertinence de son utilisation pour les étudiants.

## 7 Préparation d’un notebook pour une possible formation

Un petit *Jupyter Notebook* a été préparé pour servir de fondation à une potentielle formation. J’y ai mis une introduction aux LLMs avec des illustrations en Python, ainsi que des explications sur les biais et les limites de l’IA. Il est aussi possible d’y ajouter des exercices interactifs pour les étudiants, comme des questions à choix multiples ou des exercices de code à compléter. Une partie sur la pondération des *tokens* par les modèles à l’aide d’*interpreto* n’a pas pu être débutée, mais il serait d’intéressant de considérer cette section pour montrer aux étudiants comment les LLMs fonctionnent en interne, et l’importance de leur choix de prompts.

### 7.1 Comparaison des modèles selon les cas d’utilisation

J’ai synthétisé sur ce [tableau](#) les tâches récurrentes de développement et les modèles disponibles sur Github Copilot les plus appropriés. Pour cela j’ai utilisé cette [page de documentation](#) rédigée par Github et cet [article](#) du blog Github.

## 8 Conclusions et pistes d’exploration

À ce stade de la mission, l’avancée est comme suit:

- ☑ Étude pratique sur la pertinence de l’utilisation de l’IA dans des cas d’utilisation spécifiques
  - ☑ Cas simple avec le jeu des allumettes
  - ☑ Première évaluation d’un cas plus complexe avec l’IDM
- [ ] Finalisation de l’étude de cas IDM
  - ☐ Tests sur les TPs restants
  - ☐ Retour sur la première conclusion du rapport
- ☐ Préparation d’un notebook pour une formation
  - ☑ Introduction aux LLMs
    - Discussion sur les biais inhérents aux LLMs
  - ☑ Comparaison des modèles de langage selon les tâches

#### □ Pondération des tokens avec interpreto

Ca a été une mission plutôt enrichissante à effectuer avec beaucoup de connaissances sur l'IA Générative acquises ou renforcées. Ca a aussi été une bonne introduction à la recherche. Sur une note plus personnelle, j'ai pu prendre une certaine avance sur la matière IDM, ce qui me donnera une petite longueur d'avance pour la 2ème année. Mais j'ai aussi ressenti durant le stage que c'est un travail difficile à réaliser seul, notamment parce qu'avoir d'autres esprits sur le même sujet auraient permis d'étayer les réflexions concernant l'étude de cas, et aussi car en ce qui concerne la résolution d'un blocage, c'est souvent compliqué de trouver quelle tâche poursuivre quand autant semble être à faire en même temps.

Enfin, la conclusion actuelle vis-à-vis de l'utilisation de l'IA dans l'apprentissage est que les LLMs peuvent être un outil très puissant pour l'étudiant notamment dans la production d'un projet relativement simple avec des consignes claires qu'il comprend et connaît parfaitement. Cependant, pour une tâche nouvelle et complexe, l'apport positif de l'IA sur l'enrichissement individuel de l'étudiant est plus discutable. En effet, l'IA peut aider à la compréhension spontanée d'un concept ou d'une syntaxe ainsi que poser des bases de code exploitables, mais combiné à son potentiel d'erreurs, sa tendance à parfois mâcher le travail de l'utilisateur et son incapacité à comprendre des sujets trop complexes font pencher la balance vers l'idée qu'il vaut mieux déjà prendre en main le gros du sujet seul avant de se tourner vers l'IA pour la co-cognition.

En clair, il a été évident pour chaque test que sans connaissance préalable du sujet, l'IA peut même s'avérer contre-productive.