# Intégration de Power BI et des LLMs pour une analyse intelligente des données

Mini-Projet : Business Intelligence

Zakaria FADILI  Abderrahim HOUBBADI  Anas ROUI

**Encadré par :** Pr. Lamrani


**Ecole Mohammadia d'Ingénieurs**
Département G.MIS

8 décembre 2025

# Plan de la présentation

PARTIE I

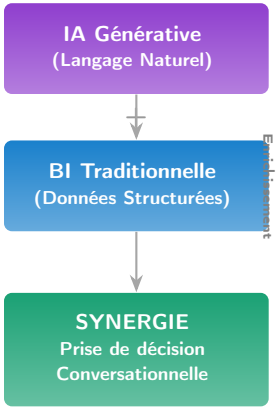
# Contexte, Problématique et Données

# Contexte : L'évolution de la BI

- **BI Traditionnelle (Power BI) :**
  - Analyse descriptive fiable et reporting structuré.
  - Modélisation robuste pour un suivi clair des indicateurs.
- **IA Générative (LLMs) :**
  - Interrogation en langage naturel.
  - Production d'insights contextualisés.
- **Synergie :**

**Vision du projet**

Ajouter une intelligence conversationnelle qui enrichit la BI et facilite la prise de décision.



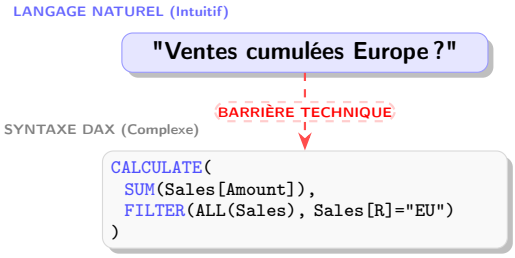
# Problématique : La barrière de l'abstraction

## Fossé Sémantique (Semantic Gap)

La complexité du **DAX** déconnecte l'expert technique (IT) du besoin métier immédiat.

**Frictions opérationnelles :**

- **High Latency (Time-to-Insight) :**
  Tout nouveau KPI impose un cycle de dév. complet.
- **Rigidité de l'Ad-hoc :**
  Impossibilité de "Slice & Dice" non modélisé.
- **Dette Technique :**
  Maintenance lourde des mesures imbriquées ('CALCULATE' complexes).



# Objectifs du Projet

L'objectif est de concevoir un **Assistant Analytique Intelligent** pour nos données de ventes.

1. **Interrogation en Langage Naturel (NL-to-DAX) :**
   - Permettre à l'utilisateur de poser des questions simples (ex : "Quel est le profit total ?").
2. **Génération dynamique de mesures :**
   - Utiliser un LLM (GPT via OpenAI API) pour traduire la question en formule DAX valide.
3. **Interface Intégrée :**
   - Développer une application Web (Interface Chatbot) connectée au modèle de données.

*Cas d'étude : Données de ventes (Sales Report) - Tables Orders & Details.*

# Data Pipeline : Ingestion et Transformation (ETL)

**1. Source & Ingestion :**

- Données brutes issues d'un repo GitHub (CSV).
- Architecture **Master-Detail (Entête-Lignes)** :
  - `Orders` (Entêtes, Client, Location).
  - `Details` (Lignes produits, Métriques financières).

**Stack Technique**

**Power Query (M Language)**

**2. Transformation & Validation (Pre-Modeling) :**

- **Casting (Typage fort) :** Conversion explicite des colonnes (ex : `Int64` pour *Amount*, `Date` pour *Order Date*) via `Table.TransformColumnTypes`.
- **Data Profiling :** Validation de la qualité des données (0% d'erreurs, 0% vide visibles sur les jauges).

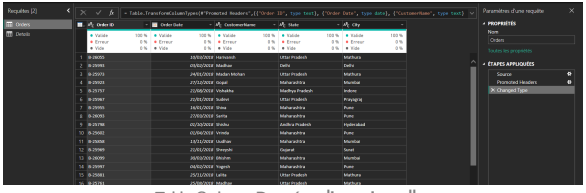


Table Orders : **Données dimensionnelles**

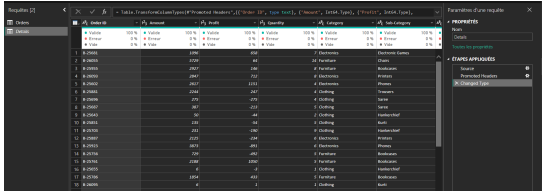


Table Details : **Données factuelles**

# Structure des Données

Nous avons structuré le modèle autour de deux tables principales :

## Table 1 : Orders (Dimension)

Contient les informations contextuelles de la commande.

- **Clé Primaire :** Order ID
- **Temporel :** Order Date
- **Client :** CustomerName
- **Géographie :** State, City

## Table 2 : Details (Faits)

Contient les métriques quantitatives et le détail des produits.

- **Clé Étrangère :** Order ID
- **Métriques :** Amount, Profit, Quantity
- **Produit :** Category, Sub-Category

# Modélisation Relationnelle : Architecture des Données

Une modélisation rigoureuse est la condition *sine qua non* pour la performance du moteur VertiPaq et la précision de l'IA.

**Spécifications de la relation :**

- **Topologie :** Entête-Lignes (Master-Detail).
- **Cardinalité :** `1:*` (One-to-Many).
- **Propagation du filtre :** Unique (`Orders` filtre `Details`).
- **Clé de jointure :** `Order ID` (Clé primaire ↔ Clé étrangère).

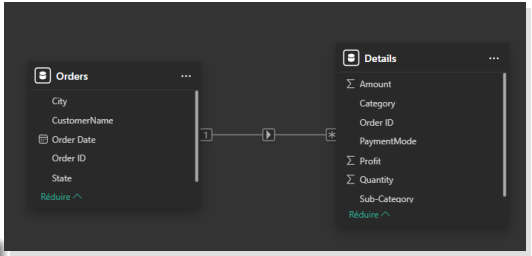



Figure : Schéma en étoile simplifié

## Impact sur le moteur IA (LLM)

Cette structure normalisée réduit l'**ambiguïté sémantique**. Elle permet au LLM de distinguer clairement les *Dimensions* (Axes d'analyse) des *Faits* (Mesures agrégables), minimisant ainsi les hallucinations lors de la génération de DAX.

# Mesures de base implémentées

Avant l'automatisation par l'IA, nous avons défini les mesures fondamentales pour valider le rapport.

**Indicateurs de Performance (KPIs) :**

- **Total Sales** : Somme des montants.
- **Total Profit** : Somme des profits.
- **Total Quantity** : Volume des ventes.
- **Profit Margin** : Ratio Profit / Ventes.

## Exemple de syntaxe DAX

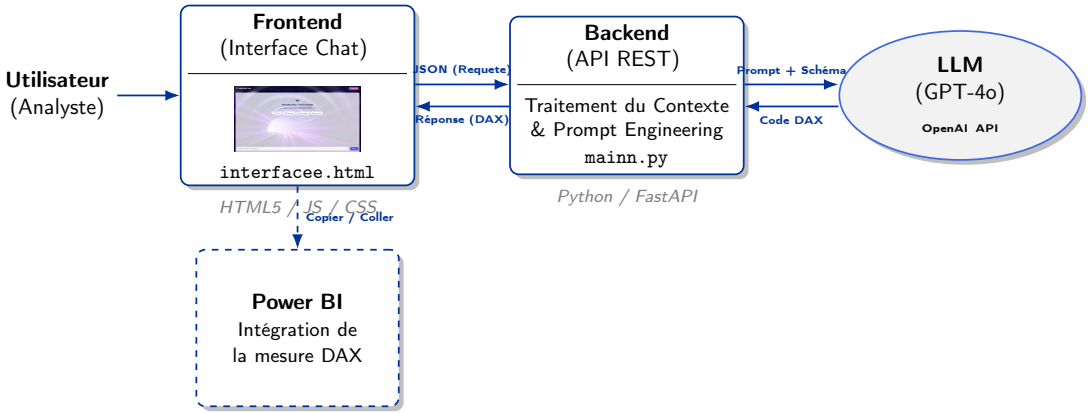
```
Total Sales =
SUM('Details'[Amount])

Profit Margin =
DIVIDE(
    [Total Profit],
    [Total Sales],
    0
)
```

PARTIE II

# Architecture Technique et Développement Backend

# Architecture de la Solution

Notre solution repose sur une architecture 3-tiers modulaire :




*Flux de données : L'utilisateur interroge l'interface, le backend enrichit la question avec le schéma de données, et l'IA génère la formule.*

# Choix Technologique : Pourquoi FastAPI ?

Pour orchestrer les échanges entre l'utilisateur et le modèle, nous avons choisi **FastAPI**, un framework Python moderne et performant.

## Avantages clés

- **Performance (ASGI) :** Utilisation d'Uvicorn pour une gestion asynchrone (non-bloquante), cruciale lors des appels API externes (OpenAI).
- **Typage Strict :** Basé sur *Python Type Hints*, réduisant les bugs de runtime.
- **Documentation Auto :** Génération automatique de Swagger UI pour tester nos endpoints.

## Rôle dans le projet

Il agit comme un *Middleware* intelligent qui sécurise et enrichit les requêtes avant de les envoyer au LLM.

# Validation des Données : Modèles Pydantic

Une API robuste commence par une validation stricte des entrées. Nous utilisons **Pydantic** pour structurer les requêtes.

**Structure de la requête entrante (`ChatRequest`) :**

```
class Message(BaseModel):
    role: str      # "user" ou "assistant"
    content: str   # Le texte du message

class ChatRequest(BaseModel):
    prompt: str
    # Historique pour le contexte conversationnel
    history: Optional[List[Message]] = []
    # Possibilité de surcharger le contexte système
    system_instruction: Optional[str] = None
```

- **prompt** : La question actuelle (ex : "Calcule le profit").
- **history** : Liste des 6 derniers échanges pour permettre le suivi ("Et pour 2018 ?").
- **system_instruction** : Le schéma DAX injecté dynamiquement.

# Workflow de l'Endpoint /chat

Voici le cycle de vie d'une requête au sein de la fonction `generate_response` :

1. **Réception & Validation :** FastAPI vérifie que le JSON respecte le schéma `ChatRequest`.
2. **Injection de Contexte :** Si le frontend n'envoie pas de contexte spécifique, le backend charge le `SALES_SCHEMA` par défaut (définition des tables Orders/Details).
3. **Appel Modulaire :** Délégation au module `LLM1.py`.
4. **Réponse :** Renvoi d'un objet JSON propre contenant uniquement le code DAX ou l'explication.

## Extrait du code d'orchestration

```
# Détermination du contexte système
context = request.system_instruction if request.system_instruction else SALES_SCHEMA

# Appel au module IA avec une température basse (0.2) pour la rigueur
reply = call_llm(
    user_prompt=request.prompt,
    system_context=context, ...
)
```

# Ingénierie de Prompt : L'Injection de Schéma

Le modèle GPT est entraîné sur du texte généraliste. Pour qu'il agisse comme un expert de **nos** données, nous utilisons une technique d'**Injection de Contexte** via le *System Prompt*.

**Contenu de la variable `SALES_SCHEMA` :**

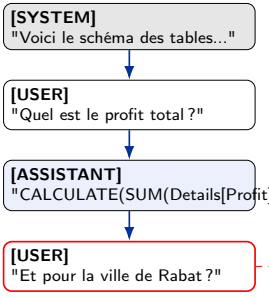
1. **Définition du Rôle :** "Tu es un expert Power BI  DAX."
2. **Structure des Tables :** Liste exhaustive des colonnes pour éviter les hallucinations (ex : inventer une colonne "Revenue" alors qu'on a "Amount").
3. **Relations :** Précision de la jointure $1 \rightarrow *$ sur `[Order ID]`.
4. **Contraintes Syntaxiques :** "Utilise DIVIDE()", "Pas de commentaires".

## Exemple de Prompt Système

```
--- RÈGLES DAX ---
1. Utilise la syntaxe anglaise.
2. Pour les ratios, utilise DIVIDE(n, d, 0).
3. Table 'Details': [Amount], [Profit]...
4. Table 'Orders': [City], [Date]...
RELATION :
'Orders'[Order ID] <-> 'Details'[Order ID]
```

# Gestion de la Conversation (Mémoire)

Les APIs LLM sont "stateless" (sans état) : elles oublient tout après chaque requête. Pour simuler une conversation continue, nous gérons l'historique côté Backend.




**Le défi :**
L'utilisateur ne répète pas "le profit".
Grâce à l'historique complet envoyé à l'API, le LLM comprend le contexte implicite.

**Optimisation :** Nous limitons l'historique aux **6 derniers échanges** pour contrôler la consommation de tokens et la latence.

# Configuration du Modèle (Hyperparamètres)

L'appel à l'API OpenAI dans `LLM1.py` est configuré spécifiquement pour la génération de code.

## Code d'appel API

```
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=messages,
    temperature=0.2, # Paramètre critique
    max_tokens=1500
)
```

**Pourquoi ces choix ?**

- **Modèle :** `gpt-4o-mini` offre le meilleur compromis Vitesse / Coût / Raisonnement logique.
- **Temperature = 0.2 :**
  - Valeur proche de 0 = Déterministe.
  - Valeur proche de 1 = Créatif.

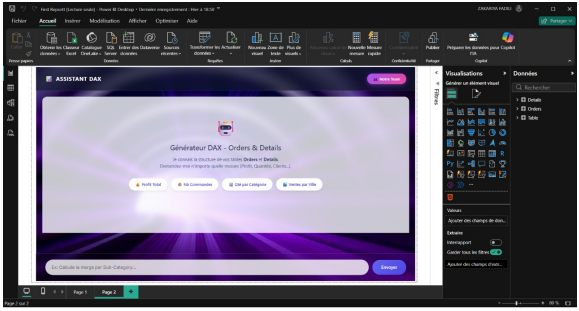
  Pour du DAX, nous voulons une réponse rigoureuse et reproductible, pas d'improvisation.

# L'Interface Utilisateur (Frontend)

Nous avons développé une interface Web moderne (`interfacee.html`) pour rendre l'outil accessible.

**Caractéristiques UX/UI :**

- **Design "Glassmorphism"** : Esthétique moderne et professionnelle.
- **Suggestions Rapides** : Boutons pour les requêtes fréquentes (Profit, Quantité).
- **Formatage du Code** : Affichage propre des formules DAX générées.
- **Indicateurs** : Statut "Écrit..." pour l'attente de réponse.



PARTIE III

# Démonstration, Résultats et Conclusion

# Scénario de Test : Du Langage Naturel au DAX

## Objectif de la démo

Démontrer la capacité de l'IA à **traduire fidèlement** une question en langage naturel (Français) en code technique (DAX), en respectant la structure de nos données.

- **Cas 1 (Simple) :**
  Calcul de performance globale (*Total Profit*).
- **Cas 2 (Complexe - Relationnel) :**
  Filtrage croisé entre Tables (*Ventes à Mumbai*).
- **Cas 3 (Analytique) :**
  Analyse de contribution (*Ratio de la sous-catégorie "Printers"*).

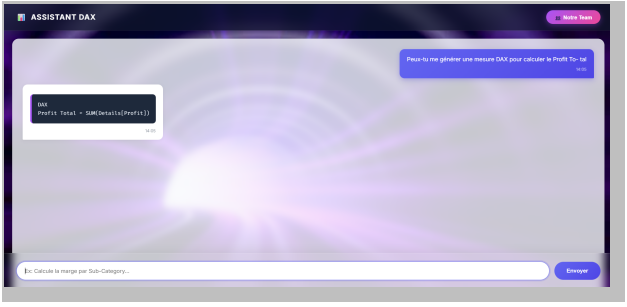
# Test 1 : Requête Simple (Agrégation)

**Question posée :**

*"Peux-tu me générer une mesure DAX pour calculer le Profit Total ?"*

**Analyse :**

- Détection du mot clé "Profit".
- Choix de la fonction `SUM`.

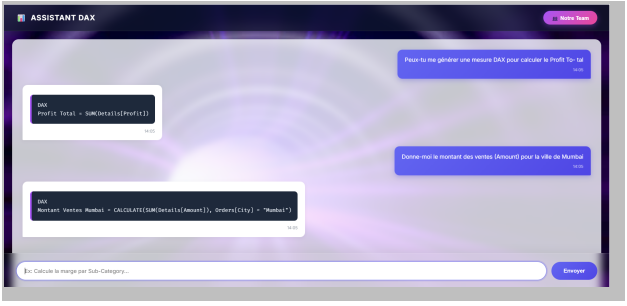


# Test 2 : Requête Relationnelle (Cross-Table)

**Question :** *"Donne-moi le montant des ventes pour la ville de Mumbai."*

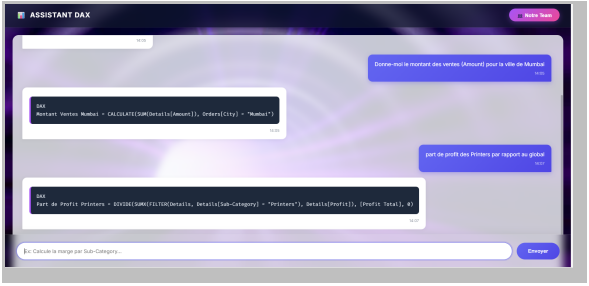
**Interprétation Technique :** C'est la preuve que l'IA comprend le **Modèle en Étoile** :

- Elle cherche la valeur à sommer (`Amount`) dans la table de faits *Details*.
- Elle cherche le filtre (`City`) dans la table de dimension *Orders*.
- Elle utilise `CALCULATE`, seule fonction capable de modifier le contexte de filtre pour croiser ces tables.



# Test 3 : Analyse Avancée (Validation Business)

**Question :** *"Part de profit des Printers par rapport au global."*




**Analyse du code généré :**

L'IA a traduit une intention métier en algorithme sécurisé :

- **Numérateur :** Elle utilise `SUMX` et `FILTER` pour isoler ligne par ligne uniquement les "Printers" dans la table *Details*.
- **Dénominateur :** Elle réutilise le concept global de `[Profit Total]`.
- **Sécurité :** Elle utilise `DIVIDE(..., 0)` pour gérer la division par zéro, une *"Best Practice"* en BI.

# Bilan Technique et Fonctionnel

| Avant (Sans Assistant) | Après (Avec Solution) |
|---|---|
| × Besoin de compétences techniques DAX. | ✓ Interrogation en langage naturel. |
| × Temps de développement long. | ✓ Génération instantanée. |

**Architecture**

Succès de l'intégration : **FastAPI (Backend) + OpenAI (Intelligence) + Power BI (Visuel)**.

# Défis Rencontrés

- **Prompt Engineering :**
  Difficulté à forcer l'IA à utiliser *uniquement* les colonnes existantes (`Category`, `PaymentMode`, etc.).

- **Hallucinations :**
  Risque que l'IA invente des fonctions DAX inexistantes (mitigé par le paramètre `temperature=0.2` dans le code).

- **Contexte :**
  La mémoire de conversation est limitée aux 6 derniers échanges pour optimiser les coûts API.

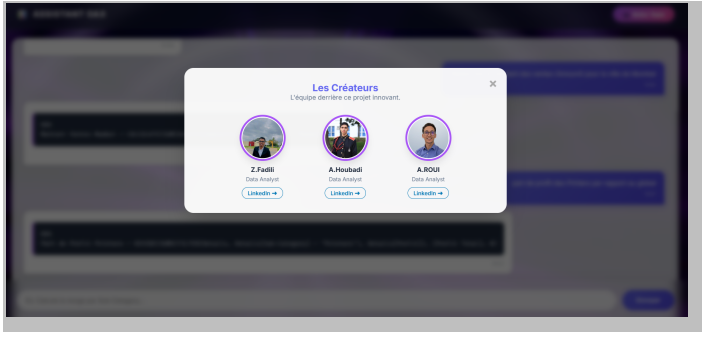
# Évolutions Futures

- **Sécurité des Données :**
  Remplacer OpenAI par un **LLM local** (ex : Llama 3) pour que les données ne sortent pas de l'entreprise (Confidentialité).

- **Intégration Totale :**
  Utiliser les **"Visuals Python"** dans Power BI pour exécuter le code DAX automatiquement sans copier-coller.

- **Déploiement :**
  Héberger l'API sur **Azure** ou un serveur **Docker** au lieu du localhost.

# Merci de votre attention.




**Avez-vous des questions ?**