Projet — Prédiction et génération pour vidéos YouTube (ex. Amixem)

> **But** : concevoir deux composantes complémentaires : une **IA prédictive** qui estime des attributs objectifs d'une future vidéo, et une **IA générative** (LLM + outils) qui propose titre et description plausibles en s'appuyant sur les prédictions et des exemples existants.

---

1) Résumé des sorties à prédire (IA prédictive)

> **Objectifs principaux** — sortir des valeurs numériques / catégorielles pour une vidéo planifiée.

- `date_de_sortie` (date) — prédire une fenêtre probable (ou un jour précis). Peut être traité comme classification sur jour/semaine/mois, ou comme régression temporelle.
- `nombre_de_vues_30j` (int) — vues cumulées 30 jours après publication (ou 7/14/90j selon granularité).
- `duree_seconds` (int) — durée en secondes.
- `sponsor_present` (bool / cat) — présence d'un sponsor dans la vidéo.
- `nombre_de_likes_30j` (int) — likes après 30 jours.
- `nombre_de_dislikes_30j` (int) — dislikes après 30 jours (si disponible).

Attributs additionnels (pertinents & pas trop lourds)

- `categorie_video` (cat) — type (challenge, vlog, test produit, let's play, compilation, etc.).
- `format` (cat) — solo / duo / équipe / invité.
- `langue` (cat) — principalement FR.
- `est_special` (bool) — vidéo spéciale (anniversaire, cross-over, événement).
- `sujet_topics` (multi-cat) — sujets/tags principaux (tech, jeux, expérience, diy...).
- `thumbnail_complexity_score` (float) — indicateur simple (ex : proportion texte sur miniature) si on extrait miniatures.
- `retention_estimee` (float) — estimation moyenne de watch-time (optionnel, utile pour vues).

> **Remarque** : pour les cibles numériques (vues/likes/dislikes), appliquer des transformations (log1p) pour atténuer l'asymétrie avant régression.

---

2) Données d'entrée (features) recommandées

Métadonnées YouTube (directes)

- `title` (texte) — historique des titres similaires.
- `description` (texte) — métadonnées existantes.
- `tags` (liste)
- `publish_timestamp` (datetime) — jour de la semaine, heure, saison.
- `duration_seconds`
- `category_id`

- `view_count_history` (série temporelle) — vues des vidéos précédentes du channel (dernier N vidéos).
- `subscriber_count` au moment de la sortie
- `engagement_rate` historique (likes/views, comments/views) pour les vidéos récentes

Features extraites / enrichies

- `delta_entre_videos` (jours depuis la dernière vidéo)
- `rolling_avg_views_N` (moyennes et écart-type sur les N dernières vidéos)
- `topic_vector` (embedding de titre + description par modèle léger)
- `sentiment_score_title` / `sentiment_score_desc`
- `title_length_chars` / `words`
- `is_weekend` / `is_holiday` (calendrier FR)
- `thumbnail_features` (si disponible) : proportion visage, texte, contraste (features simples)
- `presence_guest` (bool)

Sources de données possibles

- YouTube Data API (méta public)
- Scraping (avec prudence / respecter TOS)
- Wayback / archives pour historique
- Données manuelles (labels qualité: sponsor, format, catégorie fine)

---

3) Schéma de dataset d'exemple (ligne = 1 vidéo)

```
video_id,title,description,timestamp,duration_seconds,category_id,tags,view
s_30d,likes_30d,dislikes_30d,sponsor_present,format,topic_vector_...
,rolling_avg_views_5,days_since_last_video,thumbnail_text_ratio
```

---

4) Modèles et approches (IA prédictive)

Tâches & modèles recommandés

- Régression (vues, likes, durée si on veut) : Gradient boosting (XGBoost / LightGBM / CatBoost) ou réseaux feed-forward (si embeddings textuels lourds).
- Classification (sponsor present, catégorie) : Gradient boosting ou classification par réseau neuronal.
- Séries temporelles (date de sortie, tendances) : modèles basés sur features temporelles + time-split validation ; pour prédiction de date on peut faire classification jour/semaine ou régression sur timestamp.
- Approche end-to-end (optionnel) : modèle multi-sortie (une même NN qui prédit vues, likes, sponsor) pour tirer parti des corrélations entre cibles.

Prétraitements importants

- Log-transform `views`, `likes`, `dislikes` → prédire `log1p` puis reconvertir.
- Encodage catégoriel (target encoding pour tags/catégories fréquentes).

- Normalisation des embeddings.
- Imputation pour variables manquantes.

Métriques

- Vues / likes : MAE, RMSE sur l'échelle log et MAPE sur l'échelle originale (attention aux zéros).
- Sponsor (classification) : precision / recall / F1.
- Durée : MAE (secondes) + classification en buckets (court/moyen/long) et accuracy.

Validation

- Time-based split (train sur anciennes vidéos, test sur plus récentes) — essentiel pour éviter fuite temporelle.
- K-fold sur splits temporels (rolling-window CV).

---

5) Workflow technique — de la donnée au modèle (IA prédictive)

1. **Collecte** : récupérer métadonnées YouTube pour toutes les vidéos du channel (API / scraping). Récupérer miniatures si besoin.
2. **Labeling** : compléter labels non publics (sponsor, format, catégorie fine) via heuristiques + annotation manuelle semi-automatique.
3. **Feature engineering** : calculer rolling averages, embeddings (titre+desc), extraire jour/semaine/heure, features miniatures.
4. **Split temporel** : définir période de train / val / test.
5. **Entraînement** : entraîner modèles (XGBoost/LightGBM + NN embeddings si utile).
6. **Évaluation** : métriques listées ci-dessus.
7. **Calibration & interprétabilité** : SHAP pour expliquer drivers de vues.
8. **Packaging** : Dockerize, exposer une API interne `predict(video_candidate_metadata) -> predictions`.

---

6) Architecture & tools pour l'IA générative (LLM + tool use)

But : générer **titre** et **description** réalistes en s'appuyant sur les prédictions et sur exemples existants.

Principes :

- Le LLM (ex : Mistral via API) est utilisé comme "orchestrateur" créatif, mais il **n'écrit** qu'après avoir consulté des outils (tool use) qui fournissent exemples et données factuelles.
- Chaque outil = action distincte (ex : `tool_predict`, `tool_fetch_similar_titles`, `tool_fetch_descriptions`).

Outils (tools)

- `tool_predict(video_plan)` : renvoie prédictions (duration, views, sponsor, likes, dislikes, catégorie probable).
- `tool_find_similar(n, duration_range, views_range, topics)` : renvoie une liste de `n` vidéos existantes dont durée & vues sont similaires (métadonnées + title + id + score_similarité).

- `tool_fetch_descriptions(video_ids[])` : récupère descriptions complètes et métadonnées des vidéos fournies.
- `tool_fetch_titles(video_ids[])` : récupère titres (optionnel si `tool_find_similar` inclut titres).
- `tool_summarize_descriptions(texts[])` (optionnel) : fournit un résumé consolidé des descriptions récupérées.

Rôle du LLM (workflow interne)

1. Appel `tool_predict(video_plan)` → reçoit prédictions (duree, vues, sponsor, format).
2. LLM appelle `tool_find_similar(n=50, duration_range=±10%, views_range=±20%, topics=predicted_topics)` pour récupérer un pool de vidéos comparables.
3. LLM sélectionne (ou appelle `tool_rank`) les 3 titres les plus pertinents dans le pool ; puis acquiert leurs descriptions via `tool_fetch_descriptions`.
4. LLM construit un prompt « augmenté » contenant : la prédiction (duree, sponsor, vues), les 3 titres + descriptions, règles éditoriales (ton, longueur), contraintes SEO (mots-clés à inclure) — puis demande à Mistral de **générer** :
    - un **titre** (ou 5 variantes rangées par score de conformité) ;
    - une **description** complète (intro, timestamps suggérés, mentions sponsor, CTA, tags suggérés).
5. Post-traitement : appliquer règles automatiques (longueur, interdire mentions interdites, insertion sponsor explicitement si `sponsor_present=True`).
6. Retour final (titre, description, métadonnées générées, justification courte sur pourquoi ces choix).

---

7) Exemple de prompt structurel (pour LLM)

> **Contexte** : Données de prédiction + 3 descriptions exemples

```
Contexte: video_duration: 18m30s, predicted_views_30d: 320000, sponsor:
True (marque X), category: challenge. Ton: humoristique, francais familier.
Exemples: [titre1, description1], [titre2, description2], [titre3,
description3]
Tâche: Générer 5 titres créatifs + 1 description complète (300-500 mots)
avec: intro accrocheuse, mention sponsor en 1ère partie, 3 CTA (like,
subscribe, suivre sponsor), et 5 tags.
Contraintes: titre <= 100 caractères; pas d'affirmations factuelles non
vérifiables; rester conforme aux règles YouTube.
```

---

8) Étapes du pipeline complet (workflow global)

1. **Input** : métadonnées partielles / idée de concept (ex: "challenge nourriture piment").
2. **Prédiction** : `tool_predict` → obtient durée estimée, vues, sponsor probable, etc.
3. **Recherche d'exemples** : `tool_find_similar` avec filtres basés sur la prédiction.

4. **Extraction** : récupérer titres et descriptions des meilleures correspondances.
5. **Synthèse/Résumé** (optionnel) des descriptions récupérées.
6. **Génération** : LLM (Mistral) reçoit contexte enrichi et génère titres + description.
7. **Post-traitement & Validation** : vérifications longueur, présence / absence du sponsor, conformité.
8. **Sortie finale** : paquet {titre_candidates, description, tags_suggeres, timestamps_proposes, justification}.

---

9) Considérations pratiques & éthiques

- **Respect des données & TOS** : YouTube Data API à privilégier. Le scraping massif peut violer les conditions d'utilisation.
- **Biais & feedback loop** : si le système privilégie titres qui "marchent", il peut renforcer des patterns clickbait. Prévoir contraintes éditoriales.
- **Attribution & plagiat** : la génération doit éviter le plagiat direct des titres/descriptions ; utiliser les exemples seulement comme inspiration.
- **Vie privée** : ne pas exposer d'infos privées (données personnelles de collaborateurs/invités).
- **Transparence** : pour usage public, indiquer que le titre/description sont générés automatiquement.

---

10) Déploiement & API recommandés

- **Microservices** :
  - `predict-service` (Flask/FastAPI) : endpoint `/predict` — appelle modèle prédictif.
  - `catalogue-service` : recherche / indexation vidéos (ElasticSearch ou simple DB + embeddings).
  - `orchestrator` (LLM controller) : gère les tool calls et la conversation avec Mistral.
- **Stockage** : Postgres pour métadonnées, MinIO/S3 pour miniatures.
- **Monitoring** : suivre dérive de modèles (drift), métriques d'erreur, taux d'acceptation humaine.

---

11) Plan de validation utilisateur (MVP)

1. **MVP minimal prédictif** : prédire `duree_seconds`, `views_30d` (log-scale), `sponsor_present` ; dataset = dernières 1k vidéos.
2. **MVP génératif** : utiliser prédiction pour filtrer 200 vidéos similaires, récupérer 3 descriptions, générer 3 titres + 1 description via Mistral.
3. **Evaluation humaine** : panel (3-5 personnes) évaluent pertinence + originalité vs titres réels.

---

12) Prochaines tâches concrètes (to-do)

- Lister champs disponibles via YouTube Data API + quotas.
- Définir labels manquants à annoter (sponsor, format).
- Mettre en place pipeline d'ingestion (collector) + DB de test (1k vidéos).
- Prototyper `tool_predict` avec LightGBM sur features de base.
- Prototyper orchestrateur LLM avec appels factices aux tools.

Annexes rapides

- **Transformations cibles**: prédire `log1p(views)` puis `exp(pred)-1`.
- **Buckets durée**: court < 7min, moyen 7-20min, long > 20min (ajuster selon chaîne).
- **Taille dataset**: 1k vidéos = suffisant pour prototype, mais plus c'est mieux pour robustesse.