Détection d'opportunités d'arbitrage sur Polymarket

Encadrant : Paul Gibson
Etudiant : Paul Graczyk

18 février 2026

Table des matières

1 Introduction

Polymarket est une plateforme de marchés prédictifs sur laquelle les utilisateurs échangent des parts associées à l'issue d'événements réels (politique, sport, économie, culture). Le prix de ces parts peut être interprété comme une probabilité collective attribuée à la réalisation d'un événement.

Contrairement à un marché théoriquement parfait, Polymarket présente certaines inefficiences : pour un même marché, la somme des prix des parts YES et NO n'est pas nécessairement égale à 1. Ces incohérences ouvrent la voie à des opportunités théoriques d'arbitrage, c'est-à-dire des situations où la structure des prix viole des contraintes probabilistes élémentaires.

2 Contexte

2.1 Structure des marchés

Les marchés ouverts sur Polymarket peuvent être regroupés en deux grandes catégories : les marchés binaires et les événements multi-issues.

2.1.1 Marchés binaires

Un marché binaire ne possède que deux résolutions disjointes, généralement formulées sous la forme d'une question dont la réponse est soit "oui", soit "non" Polymarket associe deux tokens correspondants, notés **YES** et **NO**. Un exemple de marché binaire est présenté en Figure 1.

FIGURE 1 – Exemple de marché binaire.

2.1.2 Événements multi-issues

Un événement multi-issues correspond à une question comportant plusieurs résolutions disjointes possibles. Pour chacune de ces résolutions, Polymarket ouvre un **marché binaire distinct** avec des tokens **YES** et **NO**. Un exemple d'événement multi-issues est présenté en Figure 2.

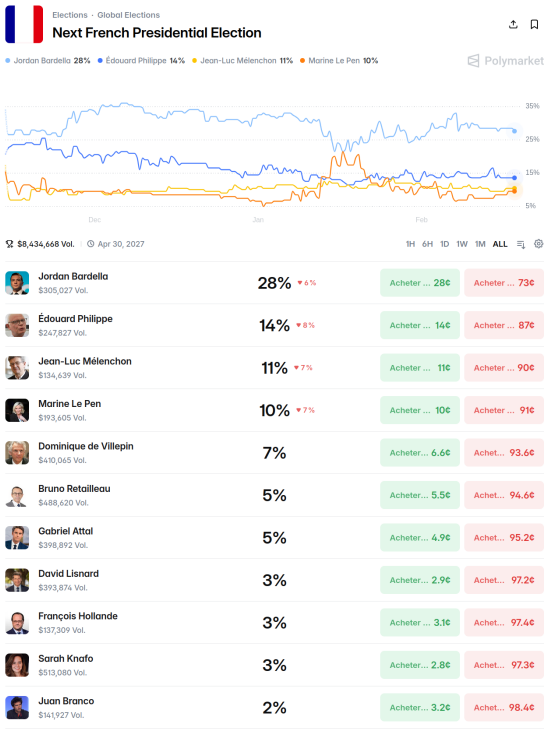

FIGURE 2 – Exemple d'événement multi-issues.

2.2 Définition d'une opportunité d'arbitrage (théorique)

Considérons un ensemble d'issues mutuellement exclusives $\{R_1, \ldots, R_K\}$ pour un même événement. On note $p_i(t)$ un prix observé à l'instant t associé à l'issue R_i (par exemple un prix de carnet choisi de manière cohérente avec l'objectif de détection).

On définit la somme des probabilités implicites :

$$S(t) = \sum_{i=1}^{K} p_i(t). \tag{1}$$

On dira qu'une opportunité théorique de type **SHORT** est détectée à l'instant t lorsque :

$$S(t) > 1 + \varepsilon. \tag{2}$$

On dira qu'une opportunité théorique de type **LONG** est détectée à l'instant t lorsque :

$$S(t) < 1 - \varepsilon. \tag{3}$$

Sinon, si $|S(t) - 1| \leq \varepsilon$, on considère qu'il n'y a pas de signal (`NONE`).

L'écart $|S(t) - 1|$ constitue une mesure de l'ampleur de l'inefficience observée à l'instant t.

En effet, dans un monde parfait, cette somme des "probabilités" devrait valoir 1. Cependant, dans notre cas les tokens YES sont soit surévalués soit sous-évalués : un arbitreur potentiel pourrait détecter cette faille et gagner $|S(t) - 1|$ avec un pari intelligent !

3 À haut niveau : le projet

Le projet consiste à développer un prototype de système d'analyse capable de détecter, de manière automatisée et continue, des opportunités théoriques d'arbitrage (au sens d'inefficience probabiliste) sur Polymarket, telles que décrites dans les sections précédentes. Le système est piloté par un module de sélection stratégique des marchés, puis applique un pipeline commun de collecte des prix, calcul de la métrique $S(t)$, détection via seuil, et enregistrement en base de données.

3.1 Stratégies de sélection des marchés

Avant d'analyser un grand nombre de marchés de façon exhaustive, il est pertinent de définir des stratégies de sélection ciblant les catégories les plus susceptibles de présenter des opportunités. Une première hypothèse (inspirée de [1]) est que certaines thématiques concentrent une part importante des inefficiences observables, notamment les marchés liés à la crypto, au sport et à la politique. Dans ce contexte, le système proposera plusieurs stratégies interchangeables, par exemple :

- `RunOnCrypto` : sélection et scan continu de marchés rattachés à la thématique crypto ;
- `RunOnPolitics` : sélection et scan continu de marchés politiques ;
- `RunOnSports` : sélection et scan continu de marchés sportifs.

En complément d'une sélection par thématique, une autre intuition consiste à prioriser les marchés présentant une forte activité ou de fortes variations de prix à un instant donné. Polymarket propose notamment une section *Breaking News* mettant en avant des marchés particulièrement dynamiques. Une stratégie `RunOnTopN` sera donc intégrée afin de sélectionner périodiquement les N marchés mis en avant dans cette section et de les analyser en priorité. On montre un exemple du "Breaking" Figure 3

Enfin, ces stratégies constituent des points de départ : elles seront comparées empiriquement à l'aide des données enregistrées (fréquence des opportunités, distribution de $|S(t) - 1|$, marchés les plus contributifs), afin d'identifier celles offrant le meilleur compromis entre coût de scan et opportunités détectées.

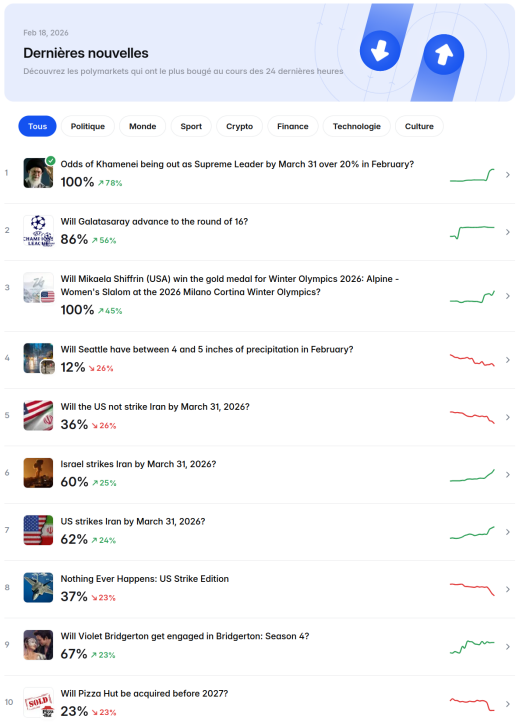

FIGURE 3 – Un exemple des marchés de Breaking sur Polymarket en février 2026

On remarque par ailleurs, Figure 3 que le Top 1, à savoir "Odds of Khamenei out by March 31 over _ in February ?" s'est résolu positivement au moment de la capture d'écran. Pourtant, quelques minutes auparavant, le pourcentage était à 60% environ : c'est un testament à la volatilité de cette section.

3.2 Pipeline d'analyse (principe général)

Indépendamment de la stratégie choisie, l'analyse suit un pipeline commun. Pour un marché donné, le système :

1. récupère les métadonnées nécessaires (identifiants, issues/outcomes, tokens) ;
2. récupère les prix observables via le carnet d'ordres ;
3. calcule $S(t)$ et l'écart $|S(t) - 1|$;
4. compare cet écart à un seuil ε (paramétrable) ;
5. si une opportunité est détectée, enregistre les informations utiles (horodatage, marché, stratégie, valeur de $S(t)$, écart, type).

Ce pipeline doit fonctionner aussi bien pour des marchés binaires que pour des événements à plusieurs issues, le point clé étant que $S(t)$ s'adapte au nombre d'issues.

3.3 Exécution continue et asynchrone

Pour éviter un fonctionnement strictement séquentiel, l'analyse est organisée de manière concurrente : plusieurs marchés peuvent être évalués simultanément. Concrètement, à chaque cycle d'exécution, la stratégie fournit un lot de marchés, puis le système lance l'analyse de ces marchés en parallèle (asynchronisme). Une fois le cycle terminé, le système attend un intervalle de temps Δt puis réitère.

Cette boucle temporelle permet :

— de maintenir un scan durable dans le temps ;
— de contrôler la fréquence d'échantillonnage via Δt ;
— de comparer la réactivité et l'efficacité des différentes stratégies.

3.4 Enregistrement et analyse a posteriori

Les opportunités détectées sont sauvegardées dans une base de données (MySQL) afin de conserver un historique exploitable. Cette persistance permet notamment :

— de retrouver les opportunités détectées (marché, date, type, écart, stratégie) ;
— de calculer des statistiques globales (fréquences, distributions, classements) ;
— de comparer empiriquement les stratégies de sélection (par exemple `RunOnTopN` vs `RunOnCrypto`).

3.5 Objectif expérimental

L'objectif expérimental du projet est d'exécuter le système de scan selon plusieurs stratégies pendant une période d'environ un mois, puis de comparer les statistiques obtenues (fréquence des opportunités, distribution de $|S(t) - 1|$, marchés les plus contributeurs) avec les résultats rapportés en 2024 dans [1]. Cette comparaison vise à étudier, au moins partiellement, l'évolution des inefficiences observables sur Polymarket entre 2024 et 2026.

4 Spécification

Cette section présente les exigences du système à développer. Le logiciel est un outil d'analyse exécuté de manière périodique, visant à détecter des incohérences probabilistes sur des marchés prédictifs. La détection repose sur la métrique $S(t)$ définie précédemment, et sur l'évaluation de l'écart $|S(t)-1|$ vis-à-vis d'un seuil paramétrable.

4.1 Fonctions à remplir par le logiciel

Les fonctions attendues sont les suivantes :

- **F1 – Sélection des marchés (stratégies).** Permettre le choix d'une stratégie de sélection de marchés (ex. top-N tendances, catégories spécifiques, multi-issues), avec paramètres configurables.
- **F2 – Collecte des données de marché.** Récupérer pour chaque marché les informations nécessaires à l'analyse : identifiants, issues (outcomes), et prix utiles issus du carnet d'ordres.
- **F3 – Calcul de la métrique.** Calculer $S(t)$ pour chaque marché analysé à l'instant t.
- **F4 – Détection d'opportunités.** Détecter une opportunité lorsque $|S(t) - 1|$ dépasse un seuil ε (paramétrable), et classifier le résultat (par exemple : `LONG`, `SHORT`, `NONE`).
- **F5 – Exécution continue et périodique.** Réitérer l'analyse en continu selon une période Δt (paramétrable).
- **F6 – Orchestration asynchrone.** Analyser plusieurs marchés en concurrence (asynchronisme), afin d'améliorer le débit de scan.
- **F7 – Persistance.** Enregistrer en base de données les opportunités détectées et les informations nécessaires à leur traçabilité (horodatage, marché, stratégie, seuil, valeur de $S(t)$, etc.).
- **F8 – Statistiques a posteriori.** Produire des statistiques à partir des données enregistrées (fréquence des opportunités, distribution des écarts, comparaison par stratégie, etc.).

4.2 Performances requises

Les performances sont définies au niveau système (objectif : scan régulier, stable et exploitable).

- **P1 – Périodicité respectée.** Le temps de traitement d'un cycle (sélection + collecte + détection + persistance) doit rester compatible avec la période Δt choisie.
- **P2 – Passage à l'échelle.** Le système doit pouvoir analyser un lot de marchés (ex. top-N) sans dégradation bloquante, grâce à l'exécution asynchrone.
- **P3 – Tolérance aux aléas.** Le système doit continuer à fonctionner

malgré des erreurs ponctuelles (API, timeouts, marchés fermés), sans arrêt global.

4.3 Contraintes de réalisation

- **C1 – Technologies.** Développement en Python. Persistance via MySQL.
- **C2 – Paramétrage.** Les paramètres suivants doivent être configurables (CLI ou fichier de configuration) : stratégie, N (si top-N), Δt, seuil ε, et éventuellement filtres de marché.
- **C3 – Dépendances externes.** Le système dépend d'interfaces de données pour (i) l'obtention des marchés à analyser et (ii) la récupération de prix.
- **C4 – Versionnement.** Le projet est suivi via un dépôt Git, accessible à l'encadrant.

4.4 Exigences de qualité

- **Q1 – Modularité.** Séparation claire des responsabilités : sélection (stratégies), collecte, détection, persistance, statistiques.
- **Q2 – Maintenabilité.** Code lisible, documenté, et structuré pour faciliter l'ajout de nouvelles stratégies ou métriques.
- **Q3 – Observabilité.** Journalisation (logs) des événements clés : cycle, erreurs, opportunités détectées, statistiques d'exécution.
- **Q4 – Testabilité.** Les composants critiques (calcul de $S(t)$, application du seuil, écriture en base, agrégations statistiques) doivent être testables indépendamment.
- **Q5 – Traçabilité.** Chaque opportunité enregistrée doit conserver l'horodatage et les paramètres utilisés (stratégie, Δt, ε) afin de permettre une analyse a posteriori.

4.5 Interfaces du logiciel

- **I1 – Interface de pilotage.** Exécution via ligne de commande (ou configuration) permettant de choisir la stratégie et ses paramètres.
- **I2 – Interfaces de données.** Modules d'accès aux données de marché (sélection des marchés / récupération des prix).
- **I3 – Interface base de données.** Module d'accès MySQL pour insérer et requêter opportunités / statistiques.

4.6 Préparation des tests de validation

Les tests (réalisés ultérieurement) permettront de valider que les exigences sont respectées :

- **TV1 – Calcul.** Vérifier que le calcul de $S(t)$ est correct sur des cas contrôlés (binaire et multi-issues).

— **TV2 – Seuil et classification.** Vérifier la détection lorsque $|S(t) - 1| > \varepsilon$ et la classification attendue (`LONG`/`SHORT`/`NONE`).
— **TV3 – Boucle Δt.** Vérifier que la réitération périodique est respectée sur une exécution prolongée.
— **TV4 – Asynchronisme.** Vérifier l'analyse concurrente d'un lot de marchés (absence de blocage global, gestion des timeouts).
— **TV5 – Persistance.** Vérifier que les enregistrements en base contiennent toutes les informations nécessaires (horodatage, paramètres, métrique).
— **TV6 – Statistiques.** Vérifier la cohérence des agrégations produites (comptages, distributions, comparaison par stratégie).

5 Conception préliminaire

Cette section présente la conception préliminaire du système : choix de solution, architecture globale, décomposition fonctionnelle, flux de données, définition des données manipulées et préparation des tests d'intégration.

5.1 Choix de la solution

Le système est conçu comme une application d'analyse exécutée en continu, pilotée par une stratégie de sélection de marchés. Les principaux choix techniques sont les suivants :

— **Langage :** Python, notamment pour la rapidité de prototypage et l'écosystème (réseau, asynchronisme, accès base de données).
— **Concurrence :** `asyncio` afin de permettre l'analyse simultanée de plusieurs marchés et d'améliorer le débit de collecte.
— **Persistance :** MySQL pour stocker les opportunités détectées et permettre des analyses statistiques a posteriori.
— **Sources de données :** une API de métadonnées (sélection/identifiants des marchés) et une API de carnet d'ordres pour récupérer les prix.

5.2 Architecture générale

La figure 4 présente l'architecture globale du système, centrée sur un moteur d'analyse exécuté périodiquement, et piloté par un module de stratégie.

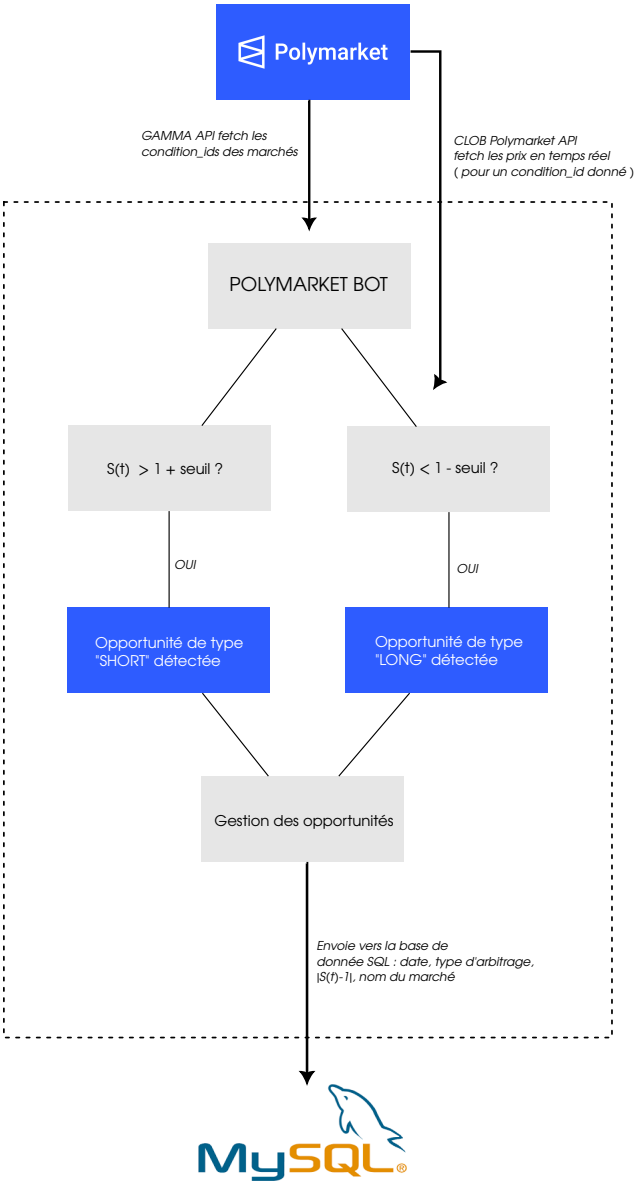

FIGURE 4 – Architecture générale suivant une certaine stratégie

5.3 Définition descendante de la structure du logiciel

La structure du logiciel est définie de manière descendante (top-down) :

1. **Niveau 0 :** un *moteur d'analyse* exécuté en continu (boucle temporelle de période Δt).

2. **Niveau 1 :** deux sous-systèmes principaux :
   — **Pilotage** : sélection des marchés via une stratégie.
   — **Analyse** : collecte des prix, calcul de métriques, détection, persistance.
3. **Niveau 2 :** modules détaillés (décrits ci-dessous).

5.4 Fonctions nominales et flux de données

5.4.1 Modules (fonctions nominales)

Les modules principaux sont :
— **Stratège (sélection)** : définit un ensemble de marchés à analyser (ex. top-N, marchés volatils, sports, multi-issues). La stratégie est paramétrable.
— **Orchestrateur** : planifie l'exécution périodique et lance l'analyse de plusieurs marchés de manière concurrente (asynchrone).
— **Collecteur de données** : récupère les informations nécessaires (identifiants, tokens, prix, état du marché) via les API.
— **Détecteur** : calcule la métrique de cohérence et décide si une opportunité est détectée (LONG/SHORT/NONE).
— **Persistance (MySQL)** : enregistre les opportunités et (optionnellement) les snapshots de prix.
— **Module statistique** : calcule des indicateurs à partir de la base (fréquences, distributions, comparaison entre stratégies).

5.4.2 Flux de données (scénario nominal)

Le scénario nominal d'un cycle d'exécution est :

1. La stratégie sélectionne un ensemble de marchés à analyser (liste d'identifiants).
2. L'orchestrateur lance la collecte pour chaque marché (en parallèle via tâches asynchrones).
3. Le collecteur récupère pour chaque marché les prix utiles (par exemple best bid/ask sur YES/NO, ou l'ensemble des issues pour un événement multi-issues).
4. Le détecteur calcule une métrique et applique un seuil pour classer le résultat :
   — opportunité `LONG` si sous-évaluation détectée ;
   — opportunité `SHORT` si sur-évaluation détectée ;
   — sinon `NONE`.
5. Si une opportunité est détectée, elle est enregistrée en base de données (avec horodatage et paramètres).
6. Le système attend Δt puis réitère sur la même sélection (ou une sélection recalculée selon la stratégie).

5.5 Définition des données (signification, type, domaine)

Les principales données manipulées par le système sont :

— **Identifiants de marché / événement :** identifiant unique (`string`).
— **Issues (outcomes) :** liste d'issues pour un événement multi-issues (`list` de `string` / `token_id`).
— **Prix :** valeurs numériques réelles dans $[0, 1]$ (`float`) issues du carnet d'ordres (achat/vente).
— **Horodatage :** date/heure de collecte (`datetime`).
— **Paramètres d'exécution :** stratégie utilisée (`string`), seuil (`float`), période Δt (`float` ou `int`).
— **Métrique d'inefficience :** valeur calculée à partir des prix (ex. somme/écart), (`float`).
— **Résultat :** type de détection `LONG`, `SHORT`, `NONE` (`enum` ou `string`).

5.6 Persistance (niveau préliminaire)

La base de données doit permettre :

— d'historiser les opportunités détectées (marché, horodatage, métrique, type, stratégie) ;
— de permettre des requêtes statistiques (fréquence par marché, par stratégie, distribution des écarts).

Une première modélisation simple peut reposer sur les entités *Market*, *Opportunity* et (optionnellement) *PriceSnapshot*.

5.7 Préparation des tests d'intégration

Les tests d'intégration (préparés à ce stade) viseront à vérifier l'interopérabilité des modules :

— **TI1 – Pipeline complet** : stratégie → collecte → détection → insertion en base.
— **TI2 – Asynchronisme** : analyse concurrente d'un lot de marchés (absence de blocage global, respect des timeouts).
— **TI3 – Robustesse API** : comportement attendu en cas d'erreur réseau (retry/backoff, logs, poursuite du scan).
— **TI4 – Persistance** : vérification que les champs attendus sont bien insérés (horodatage, stratégie, seuil, métrique).
— **TI5 – Statistiques** : cohérence des agrégations produites à partir des données stockées (comptages, distributions, comparaisons par stratégie).

6 Plan de charge

Activité	**Charge (%)**	**Contenu**
Spécification & cadrage	15%	Rédaction du livrable 1, clarification du périmètre, définition des exigences et des interfaces.
Conception (préliminaire & détaillée)	15%	Architecture globale, décomposition en modules, conception des stratégies, schéma de données (DB).
Développement	45%	Implémentation des modules (stratégies, collecte, calcul de $S(t)$, détection, orchestration asynchrone), persistance MySQL, module statistique.
Tests & validation	15%	Tests unitaires (calculs, seuils), tests d'intégration (pipeline complet), robustesse (timeouts, erreurs API), validation sur des lots de marchés.
Documentation & restitution	10%	Rédaction du rapport final, préparation de la soutenance, mise en forme des résultats et figures.

TABLE 1 – Plan de charge prévisionnel (projet solo).

7 Sources

[1] Oriol Saguillo, Vahid Ghafouri, Lucianna Kiffer, Guillermo Suarez-Tangil, *Unravelling the Probabilistic Forest : Arbitrage in Prediction Markets*, arXiv :2508.03474, 2025. DOI : 10.48550/arXiv.2508.03474.
`https://arxiv.org/abs/2508.03474`
(PDF : `https://arxiv.org/pdf/2508.03474`)

[2] Polymarket, *Polymarket (site officiel).*
`https://polymarket.com/`

[3] Polymarket Documentation, *Gamma Markets API — Overview.*
`https://docs.polymarket.com/developers/gamma-markets-api/overview`

[4] Polymarket Documentation, *CLOB API — Introduction.*
`https://docs.polymarket.com/developers/CLOB/introduction`