Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Polymarkets Hybrid Exchange System and Prediction Market Arbitrage

Bachelor's Thesis

Mario Cattaneo

mcattane@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Prof. Dr. Lucianna Kiffer, Oriol Saguillo
Prof. Dr. Roger Wattenhofer

February 2, 2026

Acknowledgements

I want to thank Prof. Dr. Lucianna Kiffer and Oriol Saguillo from IMDEA Networks for giving me the opportunity to do my thesis under their supervision and taking the time to meet me almost every week and guiding me throughout the process, as well as providing me access to their server for running my data collection.

Abstract

todo

Contents

Chapter 1

Introduction

1.1 Speculation

Classifying and regulating speculation on uncertain events has historically been challenging. A common classification hinges on the purpose of the speculation: whether it serves to hedge risk and provide insurance, to enable price discovery for information aggregation and forecasting, or to function as entertainment akin to gambling. However, in section 1.1.1 we present cases that challenge this classification and the associated regulatory approaches.

1.1.1 Case Studies

The doctrine of *Insurable Interest*, originating in 18th-century England, initially treated wagers made without pre-existing risk exposure as falling within the scope of illegitimate gambling [1]. This legal concept arose from concerns about fraudulent activity, such as the intentional destruction of insured property, and was intended to distinguish insurance from gambling by requiring a policyholder to have a financial stake in the insured item or life.

However, the "Bucket Shops" of the early 20th century in the United States, which allowed customers to wager on stock prices without ownership, prompted courts to adopt doctrines that treated such transactions as enforceable under certain conditions. These establishments operated as off-track betting parlors for stocks and commodities, enabling public participation in financial speculation without the transfer of underlying securities. In *Board of Trade v. Christie Grain* (1905), the Supreme Court recognized that a market's continuous quotations and the contractual structure of exchange dealings could justify treating functionally similar transactions as part of a legitimate market; as the Court observed, "a collection of information, otherwise entitled to protection, does not cease to be so because it concerns illegal acts" [2].

Conversely, the *Onion Futures Act of 1958* prohibited trading in onion futures in the U.S. following a substantial price collapse attributed to speculative activity,

illustrating that markets otherwise regarded as legitimate can be restricted if their harms are judged to outweigh their utility [3]. Congress enacted the Act after market manipulation attributed to traders Vincent Kosuga and Sam Siegel, who in 1955 acquired dominant positions in the onion market and contributed to sharp price declines that harmed producers. The episode suggested that certain market structures; for example, those for perishable agricultural products with limited storability and seasonal production may be particularly vulnerable to manipulation, which led Congress to impose a commodity specific restriction on onion futures trading [4].

1.1.2 Market Integrity

This regulatory ambiguity can undermine market integrity through two primary risks: direct manipulation and exploitation.

A moral hazard arises when market participants can directly influence an event's outcome. Match-fixing in sports provides an illustrative example: a boxer might have an economic incentive to lose deliberately if betting returns on their defeat exceed the prize for winning.

A related issue is asymmetric information. Not all informational advantages are malicious; some are an unavoidable consequence of information diffusion. However, the asymmetry becomes exploitative, potentially amounting to insider trading, when it arises from privileged access that allows a few actors to profit from unfairly obtained information without altering the outcome.

Both risks extend to more severe scenarios, from corporate managers taking positions that benefit from a project failure they can influence to insiders profiting from foreknowledge of military actions.

1.1.3 Modern Regulation

To mitigate these risks, regulators can prohibit speculation on events that present particularly high moral-hazard or exploitation risks. For example, the U.S. Commodity Futures Trading Commission (CFTC) implements 17 C.F.R. § 40.11, which provides that a registered entity "shall not list for trading or accept for clearing" contracts that, among other things, "involve, relate to, or reference terrorism, assassination, war, gaming, or an activity that is unlawful under any State or Federal law" and permits the Commission to identify similar activities it deems contrary to the public interest [5]. The rule has been central to oversight and review of event contracts: the CFTC has used §40.11 procedures in reviews of election- and political-event submissions, and in enforcement or review proceedings platforms such as Polymarket have restricted access for U.S. users to certain event contracts [6]. The rule has also been a focal point in litigation, including Kalshi's submissions and related judicial proceedings [7].

1.2 Prediction Market Exchanges

Polymarket and Kalshi are two exchanges popularizing prediction markets and making them accessible globally. A prediction market is derived from the outcomes of a real-world event. On both exchanges, a prediction market is exclusively designed for binary events with YES and NO shares, which are quoted in USD, or on Polymarket, more specifically, USDC, a USD-backed ERC-20 token managed by Coinbase through a reserve such that its value is identical to USD. An example is the event "Will the Republican Party win the House in 2026?", which is hosted on both exchanges. A position in a prediction market refers to a specific quantity of YES or NO shares held. The process of deciding the outcome of the underlying event and assigning terminal values for a YES and NO share is called resolution, and once the values are determined, the market is deemed resolved.

Kalshi specifies an explicit expiration date and central resolution source on its legally binding trading contracts for the YES and NO shares, such that a market is guaranteed to resolve to either YES or NO unambiguously upon expiration. If the market resolves to YES, then every YES share has a value of 1 USD, while every NO share has a value of 0 USD. Otherwise, the market resolves to NO and every NO share has the value of 1 USD, while every YES share has the value of 0 USD [8].

Polymarket, on the other hand, does not provide legally binding trading contracts, but uses the Gnosis Conditional Tokens contract (CFT) to define the YES and NO shares as ERC-1155 tokens and to enforce their combined value at 1 USDC, and it uses UMA's optimistic oracle framework for resolution 4.2. In this system, a prediction can resolve to UNKNOWN, rendering the value of YES and NO shares 0.5 USDC each 4.3.

In both cases, the combined value of a YES and NO share for a prediction is exactly 1 USD at all times, as enforced by the trading contract on Kalshi and by the CFT smart contract on Polymarket 4.1. This does not hold for the market price of the YES and NO shares on the exchange, which are determined by a Central Limit Order Book (CLOB) on both exchanges **??**. This is what makes arbitrage possible, as specified in 6.3.

Events with more than two discrete outcomes can be constructed using multiple prediction markets. For example, the event "Which party will win the House in 2026?" is decomposed into the prediction markets for the questions "Will the Democratic Party win the House in 2026?" and "Will the Republican Party win the House in 2026?". This provides powerful information aggregation capabilities built on top of the information efficiency of markets. A common pattern on both exchanges is to define an event with more than two possible discrete outcomes exhaustively, using a prediction market for all mutually exclusive outcomes. The event "Which party will win the House in 2026?" is an example of this. Polymar-

ket labels these as *Negrisk* and provides a full suite of smart contracts dedicated to these events, including a distinct settlement smart contract deployment *Negrisk CTFE* and an atomic swap operation for converting a set of mutually exclusive outcome positions to the complement position of the complement outcomes 4.5.

1.2.1 Growth

Over 2025, Kalshi's trading volume grew on average 24% weekly with a variance of 1.45 [9], while Polymarket's grew on average 31% weekly with a variance of 2.82 [10].

To put these figures into perspective, we can use the benchmarks for early-stage companies popularized by Paul Graham of Y Combinator. A weekly growth rate of 5–7% is considered good, while 10% is exceptional [11].

They fundamentally operate differently: Kalshi is fully centralized and strictly adheres to CFTC regulations, whilst Polymarket operates on a decentralized model using smart contracts on the Polygon network.

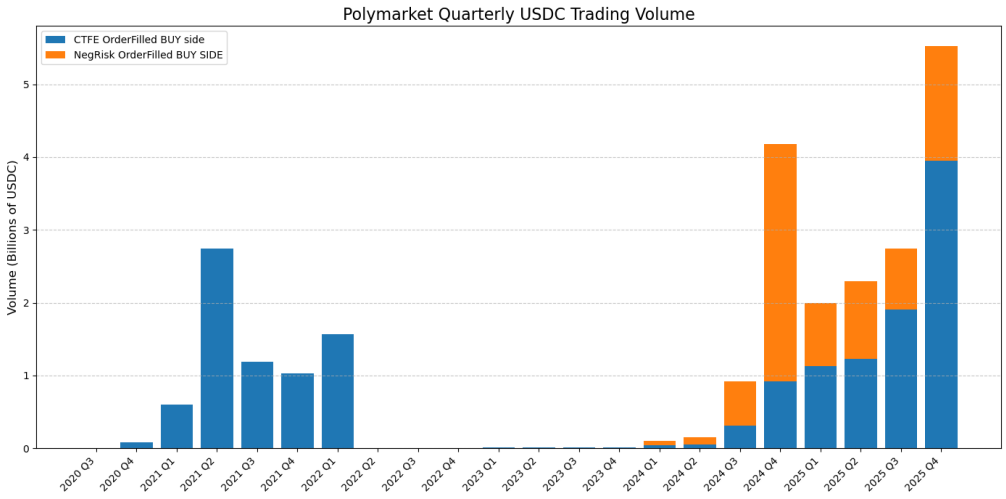

Figure 1.1: Quarterly trade volume on Polymarket. Source: Author's own compilation using Dune Analytics (https://dune.com/queries/6298071).

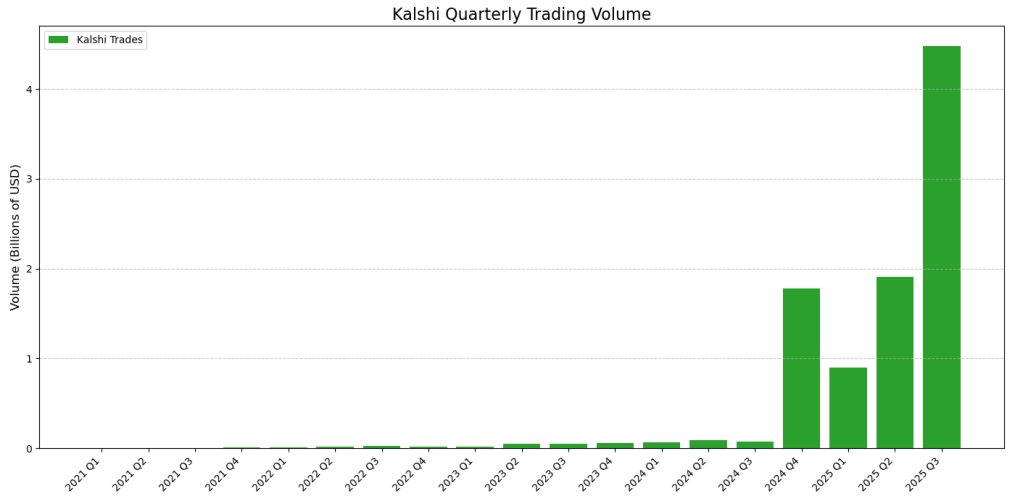

Figure 1.2: Quarterly trade volume on Kalshi. (https://tokenterminal.com/explorer/projects/kalshi/metrics/trading-volume?interval=max&granularity=quarter).

The figures 1.1 and 1.2 illustrate the quarterly USD trading volume. For Polymarket we additionally can distinguish between the trading volume on the Negrisk CTFE and Basic CTFE 4.4.

1.2.2 US Market

The immediate flat line in Polymarket's trading volume begins immediately after the CFTC action in January 2022. Kalshi operated exclusively within the U.S. until October 2025. This includes the first three quarters of 2025 and the last quarter of 2024 in figure 1.2.

The 47th U.S. presidential election was a major event on both exchanges. There have been various studies comparing the forecasting ability and information efficiency of the prediction markets on Kalshi and Polymarket for this event, specifically compared to other forecasting methods like New York Times polls [12, 13]. Besides that, you can clearly see the spike in trading volume in the fourth quarter of 2024. On Kalshi, markets related to politics and election accounted for roughly 1 billion USD volume in November alone [14], while on Polymarket, the main market for the presidential election accumulated a total volume of 3.5 billion USDC [15].

However, a recent study using two unsupervised network models suggests that around 18% or less of the volume traded on Polymarket is "washed" (i.e., likely orchestrated to inflate activity). One method reported that around 60%

of December 2024 volume was washed [16], which was the month after the U.S. election markets concluded and which showed an immediate decline of over 50% in total trading volume [17].

1.2.3 Accessibility

Kalshi has officially expanded to over 140 countries in October 2025 [18] and Polymarket has expressed intentions to reenter the U.S. On July 21, 2025, Polymarket acquired QCX LLC, a company that had registered as a Designated Contract Market (DCM) with the CFTC earlier that month [19]. This entity was subsequently rebranded as "Polymarket US" [20]. As of December 4, 2025, Polymarket is not yet available in the U.S. Both exchanges face regulatory headwinds in many jurisdictions. In many countries they are banned for usage by gambling authorities. This includes Switzerland, where the Swiss Gambling Supervisory Authority (GESPA) has blacklisted their DNS domains, making them irresolvable through Swiss ISPs [21].

Even with CFTC approval, Kalshi faces regulatory issues on a regional level in the U.S. An example is the Illinois Department of Financial and Professional Regulation (IDFPR) issuing a cease-and-desist order preventing Kalshi from operating in Illinois for allegedly defying state gambling regulation. Kalshi argues that national oversight by the CFTC preempts these claims. This ongoing case demonstrates that the regulatory landscape surrounding speculation remains conflicted [22].

By design, Polymarket remains widely accessible even if banned, since it does not enforce identification on its main trading platform (Know-Your-Customer, KYC). Even if KYC were enforced, Polymarket's traded base assets—the ERC-1155 tokens for YES or NO shares defined through the Gnosis Conditional Tokens contract—can also be obtained using *splitPosition*, swapped for USDC using *mergePosition*, and, after resolution, redeemed using *redeemPosition* for the value reported by Polymarket.

Chapter 2

Objective and Methodology

Polymarket is a hybrid exchange in which a proprietary off-chain CLOB serves as the market mechanism. Settlement of matched orders, enforcement of the non-legally binding trading contracts and resolution are implemented using smart contracts. For the off-chain CLOB, Polymarket also provides a separate market object with various attributes and categories that captures the off-chain lifecycle. The system's exact specification and coordination are poorly documented. Information is fragmented and often implicit, ambiguous, or even provably incorrect.

Endpoints to interact with Polymarket's off-chain state are hosted under subdomains of *polymarket.com*. However, throughout this thesis these endpoints have continuously changed and state inconsistencies were discovered.

Some events are available on both Polymarket and Kalshi. These include the events "Which party will win the House in 2026?" and the cyclical 15-minute event "Bitcoin up or down?".

2.1 Three Objectives

The first objective was to implement reliable data-collection clients for Polymarket's and Kalshi's endpoints to collect data for the events "Which party will win the House in 2026?" and the cyclical 15-minute event "Bitcoin up or down?". Due to poor documentation and discovered state inconsistencies, this also involved analyzing the data and state presented by the endpoints.

The second objective was to analyze how a prediction-market life cycle is coordinated on Polymarket's hybrid exchange system, as this is also not specified in the documentation.

The third objective was to formalize the prediction-market orderbook and use collected data to measure price slippage and arbitrage for the events "Which party will win the House in 2026?" and the cyclical 15-minute event "Bitcoin up or down?".

2.2 Methodology

Both Polymarket and Kalshi provide WebSocket endpoints to subscribe to orderbook state changes and trades, and HTTP endpoints to request the current orderbook state. To use these endpoints, the orderbooks of interest must be identified first. For this, both exchanges define market objects and provide HTTP endpoints to query historical market objects, as well as WebSocket endpoints to subscribe to market lifecycle events. Both exchanges use JSON for endpoint communication and define rate limits for request throughput. Due to Polymarket's poor documentation, a lot of important information had to be discovered through testing and response analysis. These findings, along with continuous endpoint changes and discovered state inconsistencies, resulted in multiple client iterations. The final clients are tailored for the events "Which party will win the House in 2026?" and the cyclical 15-minute event "Bitcoin up or down?", and includes strict state and response monitoring and handling. This is covered in chapter 3.

To analyze the market lifecycle, first the statically available information of on-chain smart contracts is analyzed in chapter 4. This consists of essential smart contract control flow, events emitted, and immutable contract storage, which allows important links to be made between contracts and their intended use, as well as also help identify information that can only be deduced or inferred from the emitted events. For this purpose a client for the Infura 'eth_getLogs' endpoint was implemented. These events and the historical market objects, which are fetchable through Polymarket's endpoints, are then used to analyze how a market lifecycle is coordinated in chapter 5.

In chapter 6 the orderbook for prediction markets is formalized to define how slippage and arbitrage for the events "Which party will win the House in 2026?" and the cyclical 15-minute event "Bitcoin up or down?" are measured. Then the data collected from the Polymarkets and Kalshis endpoints, and emitted events are used to analyze slippage and arbitrage by these measures.

Finally, chapter 7 discusses and interprets the findings of this thesis.

Chapter 3

Endpoints and Clients

Polymarket and Kalshi provide endpoints on subdomains of *polymarket.com*. For the purpose of our data exploration we're interested in the endpoints which allow us to later analyze the market lifecycle, order book state and trades.

3.1 Polymarket Event and Market Endpoints

Polymarket has a market object for every prediction market, with a unique market id. Additionally, every market is also part of an *event*, and every event can be associated with multiple *tags*. The *gamma-api.polymarket.com* domain hosts a handful of HTTP endpoints to query market objects, event objects or tag objects in pages. The documentation only provides an example use case a long with list of the query parameters for the request and attributes in the response with a sample request and response. The most essential query parameters are *offset* and *limit* which enable accessing objects in batches, where the limit specifies how many objects are in a requested batch and the offset specifies the starting object for that batch. This implicitly assumes that the objects are somehow ordered, with the first object starting at offset zero. Additionally, an order parameter exists, without further specification. Testing has shown that using it for almost all attributes either does not order the attribute(s) to be ordered, misses some markets or returns duplicate markets at separate offsets. Querying the event objects ordered by event id seems to work without any of the anomalies mentioned above, and has recently been added as an example use in their documentation. However, ordering the market objects by id is not possible. Additionally, unspecified and found through testing; the limit parameter can at most be 500, any value greater will simply always return 500 objects starting at the specified offset. Before December 2025, it was also not specified what the response would be if an offset was requested beyond the amount of available offsets. This was found, and recently clarified in the example use case, to be an empty JSON array.

3.1.1 Event and Market Objects

The market object is listed with over 100 attributes in the documentation. Initial testing has shown that there is a mismatch between attributes that appear in responses and those listed in the documentation, and also differs by response. For this reason all event objects, which also contain associated market and tag objects, obtainable by querying the *events* endpoint are analyzed. Since JSON is specified by a deterministic context free language, we can parse it unambiguously into a tree of nodes, where every node has a type. Now for every event object we construct this tree, where every node, except for the root, either is the child of node of type object or array. In the case that it is the child of an object, it must have a string key associated. A node is a leaf if it is an empty array or object, or of type scalar; number, string, boolean or null. For a more representive type analysis the numbers are further broken down into IEEE 754 floating point and integers, whilst the strings are divided into empty strings, only spaces, JSON encoded, standardized timestamp formats, numbers, hexadecimal, case insensitive "null", "none", "nan", "true", "false", or a *general* string otherwise. The resulting table with the type makeup and appearance percentage for every tree path can be found in the Appendix C.1 and will be referenced throughout the analytics.

3.2 Polymarket Orderbook and Trade Endpoints

Polymarket provides a *book* HTTP endpoint, to request the current orderbook state, under the *clob.polymarket.com* domain and a *market* Websocket endpoint, to receive events for state changes, under the *ws-subscriptions-clob.polymarket.com* domain, which is referred to as the *market feed*. Every prediction market is associated with two orderbooks, one for the YES outcome and one for the NO outcome. An orderbook has a side for the prices of buying the underlying outcome token, also known as *bids*, and a side for the prices of selling the underlying outcome token, also known as *asks*. The orderbooks underlying outcome *token id* or also referred to as *asset id*, uniquely identifies it and every market object contains two asset ids inside of a JSON array encoded string, under the key "clobTokenIds" C.1.

3.2.1 Book Endpoint

The specification for the book endpoint only consists of list of the request and response JSON attributes with sample for both. This endpoint returns the orderbook state objects for the requested asset ids. The documentation does not specify the amount of asset ids that can be requested or orderbooks that can be returned, nor what happens if the orderbook of an already resolved market is requested. Testing shows that the orderbooks states for requested asset ids are

returned in the order they were requested, any orderbook state object requested for an asset id, whos market is already resolved will not be in the response, and the response consists of at most 500 orderbook state objects.

The orderbook state object in the appendix **??** is from the sample Polymarket provides in its documentation. Importantly, the timestamp attribute in the response is listed as a string, but is actually a string of an integer unix millisecond timestamp in the responses.

3.2.2 Falsely Missing Orderbook State Objects

An initial client which made concurrent requests to the book endpoint, revealed an anomaly in the state provided from the endpoint, in which for some asset ids the endpoint would not return the order book state objects for one or multiple requests and then in later request include it again in the response. Essentially making order book state unavailable or seem as if its market has been resolved. Importantly, this could not have been a result from infringing the rate limit, as the endpoint would send regular bad response statuses, which was not the case here. The same set up was run for a couple of hours, but now making strictly serialized requests, and this anomaly was not detected. To investigate this further, every asset id was mapped to a *false miss* counter, which was incremented whenever the orderbook state object for requested asset id was not in the response, and if it was in the response, then the order book state object and counter are stored and this counter was set to 0.

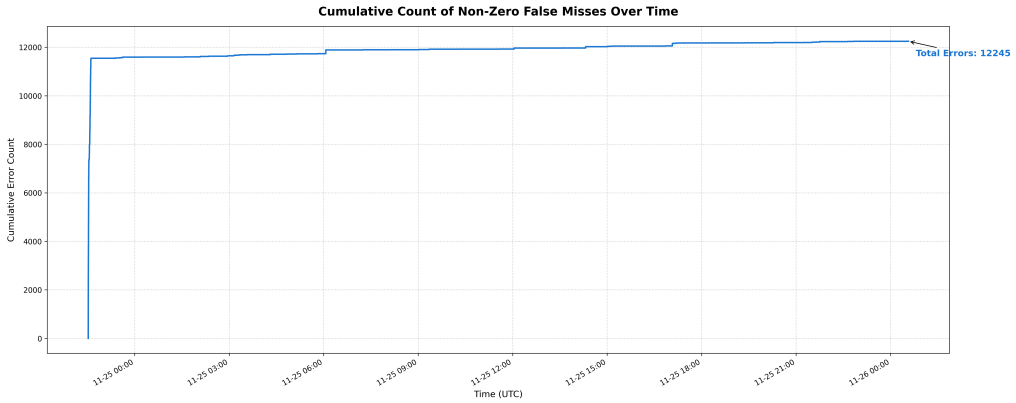

Figure 3.1: Cumulative count of falsely missed asset ids

In 3.1 we can see that initially there is a burst of false miss reports.

illustrate that initially false miss count is in rang 0-15 then it there are fewer but much larger culprits

3.2.3 Market Feed

The market feed sends out different event messages for subscribed markets. Before December 17 2025 this endpoint only allowed for subscription once on an established Websocket connection. This means that if any markets were resolved or created, a new Websocket connection had to be established to adapt for this change. This was not documented, and testing simply revealed that attempting to send more than one subscription message would stop any events from arriving. Even with a timeout of 30 seconds the server was still keeping the underlying persistant TCP and Websocket connection alive and no other messages were sent, essentially turning it into a zombie.

On December 17th dynamic subscriptions and unsubscription messages were announced through their discord server, confirming that this in fact was not possible beforehand. This was a game changer, as the prior client had to create a new Websocket connection whenever new markets were created. Another significant improvement was introduced around January 5th 2026, where life cycle updates were added to the market feed. Before this the market or events endpoint had to be scraped continuously, to check if a new market was created or an existing market was resolved. This introduced unnecessary latency and complexity. Comparing the market objects found on the long lived connection, which was continuously scraped for market life cycle changes, to a separate one time exhaustive scrape on a different connection revealed that some market objects were never returned at any offset on the market endpoint for the long lived connection. This anomaly was only measurable after 3 days.

Unlike for the book, market and event endpoint, the documented response attributes appear in every response and are of the correct type. The documented sample message can be found in the appendix **??** for the price change event, **??** for the book event, **??** for the trade event, **??** for the trade event, **??** for the new market event and **??** for the market resolved event. However, for the last trade price event, there is a missing attribute *tx_hash* of type string, which always is hexadecimal and has the length of an ethereum transaction hash. The timestamp of the market feed events are a string of an integer unix millisecond timestamp, just like the one found in the actual responses of the order book state object requests. However the fact the timestamps are only in millisecond resolution and there are no other attributes to order the events by, would render state ambiguous after a millisecond with multiple conflicting events.

3.2.4 Ambiguous Event Ordering

The most obvious conflict, is between events of the same type at the same millisecond with different resulting state. For price change events, this is the case when two or more price change events have the same timestamp, side, price and asset id, but different sizes, as then it is unclear exactly what the size is at the

given price, side and asset id after or at the timestamp. Book events are conflicting if they share the same timestamp and asset id, but not the same bids and asks, and last trade events are conflicting when they share the same timestamp, asset id, side and price, as we can then no longer differentiate between which trade got filled using which size and price. Table 3.1 illustrates that this is not just theoretically possible, but actually also happens.

Table 3.1: Bitcoin Up or Down — January 7, 11:00AM-11:15AM ET

Message Type	**Total**	**Ambiguous (%)**	**Duplicate (%)**
Asset ID: 113615...8317859			
Last Trade Prices	3,562	24 (0.67%)	0 (0.0%)
Price Changes	158,662	3'228 (2.04%)	1'336 (0.84%)
Books	6,074	0 (0.0%)	0 (0.0%)

If we additionally consider the ambiguity that arises when a price change message and book summary message share the same timestamp, i.e unclear whether the price change applied before or after the book sate summary, we find that 7'160 price changes are ambiguous.

3.3 Final Polymarket Client

The initial client was intended to subsribe to the market feed and make order book state requests for all existing unresolved markets. This involved over 30 thousand orderbooks. Since only 500 orderbooks can be requested at once, there is a clear trade off between how many orderbooks are used for requests and how frequently we can request an orderbook. Additionally, the fact that the market feed events can not unambiguously be ordered, makes requesting the actual orderbook state frequently even more valuable. For these reasons the final client was designed to use less than 500 orderbook, and request their state strictly serially, with at least 500ms between requests. The documentation specifies a rate limit of 500 requests/s, so this setup is very conservative. For the event "Which party will win the house in 2026?" there are 4 order books and for the cyclical 15 minutes "Bitcoin up or down?" there seem to be less than 100 active order books.

The final client initially scrapes the events endpoint for all currently available 15 min cyclical "Bitcoin up or down?" markets. Then subscribe to the to the market feed using these and the 4 known orderbooks from the 'Which party will win the house in 2026?" to subscribe to the market feed, and listen for newly created or resolved "Bitcoin up or down?" markets, to then subscribe or unsubscribe to. We also start making the orderbook state requests as specified above, and obviously store the price change, book, last trade price events, and orderbook state objects persistently in a database table.

3.4 Kalshi Endpoints

Kalshi provides endpoints similar to those on Polymarket. Namely the *events* endpoint for scraping any market ever created and a websocket endpoint with different channels, like *trades*, *orderbook updates* and *market lifecycle*. Markets are uniquely identified through their market_ticker. Dynamic subscription and unsubscription for the channels of interest, on the websocket endpoint, can be done using these market_tickers.

However unlike Polymarket, the events are not in the same format as on Polymarket, more importantly, Kalshi provides a single orderbook for both outcomes of a prediction market, where one side is for the YES prices and the other is for the NO prices [**?**]. The currently documented JSON format of the events collected is specifed in the appendix **??**

The JSON objects inside of the messages sent by the orderbook update and trade channel contain a unique integer sequence identifier, which can be used unambiguously order the events for a given channel and spot any missing events, since the first event after subscription has sequence identifier zero, and for all other events the sequence identifier is the sequence identifier of the event happening right before it, incremented by one. However, this does not make ordering the trade events among the orderbook events unambiguously possible, as for this we would have to rely on the timestamps alone, for which the trades only specified in seconds.

3.5 Kalshi Client

Because of the sequence identifieres provided by kalshi, there is no need to additionally request the orderbook state, as it can be used to unambiguously maintain the state of the orderbook. The Kalshi client therefor simply subsribes t

Chapter 4

Smart Contracts

Polymarket has a dedicated github account with over 90 repositories. There are many individual repositories containing the solidity source for smart contracts and their deployment status. They also have a dedicated repository which lists any deployment address that seemingly can be found elsewhere, the associated contract and a security audit, if present [23]. It is not explicitly mentioned if it is an exhaustive list, but at least looking through the entire documentation on the website and all repositories on github, does not reveal any undeclared deployment, since there is no other authorative source overriding this, it will be used as the reference of the disclosed deployments.

4.1 Conditional Tokens

The Gnosis Conditional Tokens contract (CTF) extends the ERC-1155 standard, utilizing it to construct tokens representing the outcomes of an event or condition. This enables transferring, minting, and burning multiple outcome tokens within a single transaction.

The contract defines the relationship between the value of the n outcome tokens $(O_1, \ldots, O_n)$ for an event and the ERC-20 collateral token Q as:

$$Q = \sum_{i=1}^{n} O_i \tag{4.1}$$

The *splitPositions* method atomically swaps an amount n of collateral Q for n units of each outcome token and emits the *PositionSplit* event. The *mergePositions* method performs the reverse atomic swap and emits the *PositionsMerge* event.

Before these methods can be utilized, the condition must be prepared using the *prepareConditions* method with an oracle address, a question ID, and an outcome count. This emits the *ConditionPreparation* event. Every condition is

assigned a unique identifier defined as:

$$condition_{id} := keccak(oracle \parallel question_{id} \parallel \text{outcome count}) \tag{4.2}$$

Subsequently, only the oracle can report the payouts using the *reportPayouts* method by passing an array of unsigned integers $payouts = (p_1, ..., p_n)$ for the n outcome tokens respectively. This emits the *ConditionResolution* event and assigns each outcome token O_i a value new value:

$$O_i := Q \cdot \frac{p_i}{\sum_{k=1}^{n} p_k} \tag{4.3}$$

After this assigment we still have that (4.1) holds.

Finally, after the payout is reported, anyone can redeem the outcome tokens they own for the value assigned to them in (4.3) using the *redeemPositions* method, which emits the *PayoutRedemption* event.

4.2 UMA framework

UMA designs protocols for oracles based on gametheory and deploys ethereum based contracts on various block chains like polygon. These deployments are connected to UMAs dedicated web application, which is allows users actively to participate in solving price requests.

Polymarket exclusively used UMAs OptimisticOracleV2 (OOV2) deployment up until August 2025, where the UMIP-189 governance proposal passed and UMA deployed a ManagedOptimisticOracleV2 (MOOV2) dedicated only for Polymarket usage. The ManagedOptimisticOracleV2 is based on the same protocol, but with additional access control configurations [24].

The protocol implemented in OOV2 differs based on wether the price request is set to be event based or non event based. Polymarket atomically sets the requested prices to be event based in its UmaCtfAdapter contract. Which starts with a price request specifying an identifier, timestamp, ancillary data, ERC-20 currency and proposal reward. UMA manages an AddressWhitelist for requesters which can make price requests. This emits the *RequestPrice* event. After the price was requested there are configurations that can be set this includes whether it is event based or not as well as a the proposal bond, liveness window for disputing proposals, or setting a callback for price proposals, proposal disputes and settlement. These can only be configured by the requester who made the request price. For a price request to be settled, the effective price must first be determined, this requires an initial price proposal to be made, by suggesting a price for the request identified by the requester, id, timestamp and ancillaryData. This emits the *ProposePrice* event. After a price proposal is made, the price can be disputed within the liveness period which by default is two hours again with

the same identifiers for the request. If the proposal remains undisputed during this period, then it becomes the effective price for the request and the request can be settled. Both the proposals and disputes have no access control in OOV2. If the proposal is disputed the *DisputePrice* event is emitted and then a stake based voting process starts. The contract for this is deployed on the Ethereum network and therefore involves an off chain coordinated message relayer. After the price is determined the request can be settled, to reward or slash the proposer, and disputer if existent, depending on the correctness of their proposals compared to the final outcome. This emits the *Settle* event.

The OOV2 does not specify what the prices exactly mean, so in the context of Polymarket the ancillary data has to formally specify a semantic meaning of the integer price values. Polymarket does not explicitly state this, but implicitly seems to adhere to the UMIP-107 standard, in which the requested price identifier is *YES_OR_NO_QUERY* and the ancillary data is a utf-8 encoded dictionary, which specifies the question using the *q:* key and the 4 possible prices through the *p1:*, *p2:*, *p3:* and *p4:* keys. Which specify which price is mapped to the outcomes *YES*, *NO*, *UNKNOWN* and *EARLY PROPOSAL* [25].

4.3 UmaCtfAdapter

This is polymarkets custom contract, built on top of the CFT and OOV2, which it stores as immutable references by address. It provides an *initialize* method which takes ancillary data and other UMA parameters. It initializes a market by preparing the condition in a CFT instance using itself as an oracle, $question_{id} := keccak(\text{ancillary data})$, and a hard coded outcome slot count of two. It also makes a price request with the hard coded *YES_OR_NO_QUERY* identifier, current block timestamp, ancillary data and the other parameters, and also in the same transaction sets the price request to be event based and only sets a callback for disputes.

The *initialize* method is permissionless and emits the *QuestionInitialized* event. In fact, any one can use it to make a price request through the contracts deployed address as requester, even though the OOV2 has a curated requester whitelist. The other permissionless method *resolve*, emitting the *QuestionResolved* event, can only be invoked if the markets requested price has been determined and it is not paused. It calls the OOV2 to settle and get the *int256* price and if the price is not that of the *EARLY PROPOSAL* outcome, then uses this price to report the payouts in the CTF. The *EARLY PROPOSAL* outcome is hard coded as the least *int256* value for comparison, a price of 0 is considered as *NO*, a price of 0.5 ether is considered as *UNKNOWN*, any other price by contract logic is considered *YES*. This means that even if the returned price by OOV2 is not the one specified as *YES* in the ancillary data, the outcome could be considered *YES* by contract logic. If the outcome is *NO* then the payout array

passed to the CFT is [0,1], if the outcome is *UNKNOWN* then the it is [1,1] and otherwise it is [1,0] for the outcome *YES*.

The rest of the public methods in this contract all have access of either *onlyAdmin* or *onlyOracle*, such as the *pause* method mentioned before, which prevents the resolve method from being invoked, but there others like *resolveManually* which directly reports the payouts array passed without resolution from the OOV2. Essentially providing centralized control on the market lifecycle. All of them emit events and are part of the exhaustive list in the appendix

4.4 CTFExchange

The CTFExchange is another custom contract by Polymarket it provides the *matchOrders* method for which is used for settling a taker order and the matched n maker orders and emits one *OrdersMatched* event and $n+1$ *OrderFilled* events. Since $n \geq 1$ we always have atleast 2 *Orderfilled* events for every *OrdersMatched* event. For split matches the tokens are created using *splitPosition* in the CFT. Whilst for marge matched order the outcome tokens are swapped for the collateral using *mergePositions* from the CFT. This and the other public methods in this contract all have restricted access.

A related module built on top of the CTFExchange is the FeeModule which is a proxy for the CTFExchanges *matchOrders* method. It is also used to manage trading fees. Emitting the *FeeWithdrawn* and *FeeRefunded* event.

4.4.1 Blockchain Slippage

There are plenty of case studies conducted on this matter, but the general conclusion is that market participants in decentralized market mechanisms face a tradeoff between lowering their slippage tolerance, risking the trade being aborted, or simply paying an additional trading fee for the price slippage [26].

Emerging blockchains, like Solana, implement various measures to mitigate this, such as increasing state synchronization frequency to reduce execution uncertainty, and minimizing gas fees. Lower fees make aborted transactions less expensive, allowing users to set lower slippage tolerances without financial penalty [27]. Some blockchains even centralize node management to prevent predatory transaction reordering; Polygon, for example, curates its validator set through a centralized application process and limits it to a fixed size.

However, since Polymarket orders are placed on centralized orderbook (CLOB) over HTTPS and order flow data is not publicly available through the CLOB this vector, there cannot be any maliciously induced price slippage, unless Polymarket itself manipulates the market or on chain settlements somehow leak enough

information in a timely manner. Price slippage still is possible and measurable on polymarket as illustrated in **??**.

4.5 Negrisk

The Negrisk framework builds on top of the UmaCtfAdpater to create multiple markets with mutually exclusive outcomes and thereby provide the *convertPositions* method, which enables atomically swapping a set YES shares to the complementary set of NO shares or vice versa. This is implemented in the Negrisk Adapter, but the framework also entails a Negrisk Operator, which is used by the Negrisk UmaCtfAdapter as immutable reference for the conditional tokens contract interface, but the Negrisk Operators conditions prepared method does literally nothing and its reportPayouts method does not call any other contract, but it has other public methods which relay to the adapter, which actually references the CFT. Additionally a dedicated Nergisk CTFExchange deployment exists, which is simply another instance of the CTFExchange contract.

figure for immutable deployment references

4.6 Polygon Collection

The polygon collection is tailored for the Infura eth_getLogs endpoint. Infura has a rate limit of 500 credits/s, with a request to the eth_getLogs requiring 255 credits according the documentation. However, testing showed that the actual rate limit between requests without getting rate limited is more around 1s. The rtt for a single eth_getLogs request can be over 30s long, which is why its suited for concurrent requests. Besides the request frequency, there also is limit of 10'000 events that can requested. But the request is specified using a block range. For this purpose we can use alpha estimation with a margin 500 to converge to the block density for requesting the block expected to return 9'500 events. This seems to be effective as for most events the block density does not seem to have strong variance. But the margin can be increased if some events have a greater block density variance. As a sanity check the amount of events returned have been compared to the amount of events returned to that found in dune over the same block range. The full list of collectected events can be found int the appendix B.2

Chapter 5

Market Lifecycle

The Infura client specified in section **??** collected the events from appendix **??** from the Polygon block number x, with timestamp xx, up until block number y, with timestamp yy. For the off chain market life cycle we can scrape the events endpoint, with historical event and market objects, as described in section **??**.

5.1 Market Object Timing

The seven timestamp attributes which appear consistently (over 90%) of the market objects are *createdAt*, *updatedAt*, *endDate*, *startDate*, *acceptingOrdersTimestamp*, *umaEndDate*, *closedTime*, as can be seen in appendix **??**. While their names are indicative of their meaning, no ordering or timing is specified in relationship to each other as well the on chain events. If for all ordered pairs of timestamps (T_1, T_2) we compute $P(T_1 \leq T_2)$ we see that for many pairs we have a fully consistent order where $P(T_1 \leq T_2) = 1$ or $P(T_1 \leq T_2) = 0$ in some cases there seem to be outliers such $P(T_1 \leq T_2)$ is close to 0 or 1.

In any case to obtain a well ordered subset of the timestamps above where all of the timestamps can be compared to each other by the happens after or at $\leq_\tau$ relationship, where we can set a threshold $\tau \in [0, 1]$ such that $T_1 \leq_\tau T_2 \iff P(T_1 \leq T_2) \geq 1 - \tau$. If we set $\tau = 0$ the largest well ordered subset we get is of size 2. If we set $\tau = 0.07$ we get that the largest well ordered subset is of uniquely $\{acceptingOrdersTimestamp, closedTime, createdAt, startDate, umaEndDate, updatedAt\}$ with order:

$$\begin{aligned} &createdAt \leq_0 acceptingOrdersTimestamp \leq_{0.031} startDate \\ &\leq_{0.0002} closedTime \leq_{0.0087} umaEndDate \leq_{0.0009} updatedAt \end{aligned} \tag{5.1}$$

Where the tolerance in the ordering is set to the respective minimal value such that the order holds. Importantly the minimal tolerance is not transitive, since

not every market has all timestamps and therefor comparisons vary. For example for the order above, the actual minimal tolerance τ required is 0.0598 for $createdAt \leq_{0.0598} startDate$, which is significantly larger then the transitive closure of $createdAt \leq_0 acceptingOrdersTimestamp \leq_{0.031} startDate$ would suggest. This is because all of the markets have a *startDate* and *createdAt* timestamp, but only 90.4% of markets have an *acceptingOrdersTimestamp*, as can be seen in the table C.1.

5.2 Unkown Reporting Oracles

In section 4.1 we saw that the oracle redefined for every condition preparation on the CFT, and is the only address which can report payouts for the final YES and NO share value after resolution. Which is why it is essential to analyze which oracles are being used and how they operate.

Matching conditions ids found in the market objects with the condition ids from ConditionPreparation events reveals 2 non disclosed deployment addresses with the alias *MOOV2 Adapter* and *Centralized Adapter* in B.1.

Table 5.1: Oracle Makeup from 123,319 Conditions Prepared

Alias	**All Events (%)**	**Matched Events (%)**
[?] heightMOO2 Adapter	55,176 (44.68%)	55,052 (99.78%)
Centralized Adapter	42,181 (34.16%)	39,456 (93.53%)
Negrisk Adapter	25,962 (21.02%)	25,438(97.98%)

The MOOV2 Adapter is an UmaCtfAdapter instance described in section 4.3. The MOOV2 Adapter uses the MOOV2 instead of the OOV2 deployment, which UMA specifically deployed for Polymarket as explained in 4.2. On the other hand Negrisk Adapter and Basic Adapter use the OOV2. However, no condition was prepared from the Base Adapter, which was has its own deployment repository and is also on in the dedicated deployment list as explained in **??**.

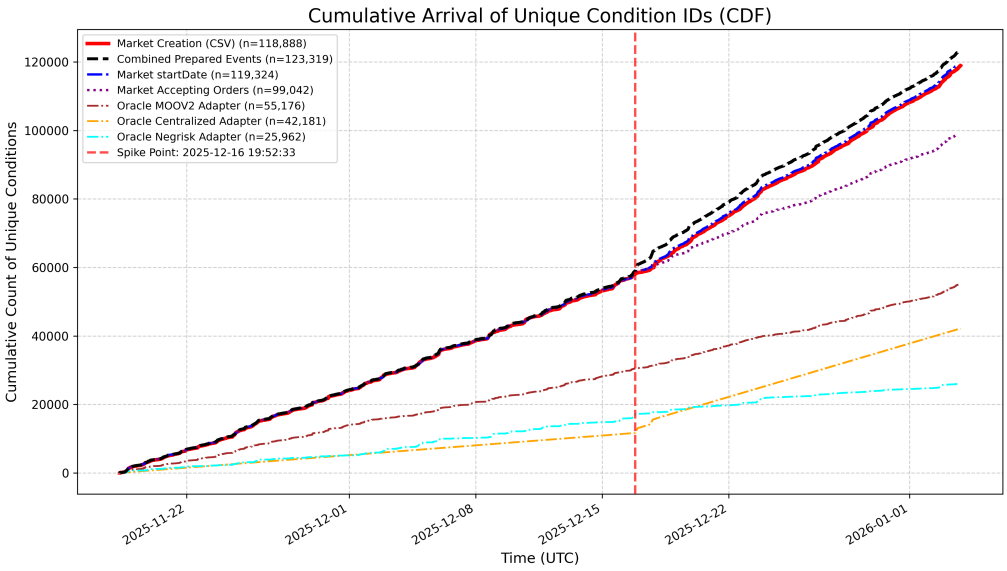

Figure 5.1: Market *createdAt* and Condition Preparation timing

The figure 5.1 shows how the market object timestamps compare to the conditions preparared using the matched oracles. The dashed red line is an interesting point, because the accepting orders time starts to diverge from the createdAt and startDate times. This also introduced a constant gap between the conditions prepared in the CFT and the markets created. At the same time the Centralized Adapter had a sudden change in the amount of conditions being prepared.

5.3 Reporting Oracles and Resolution Source

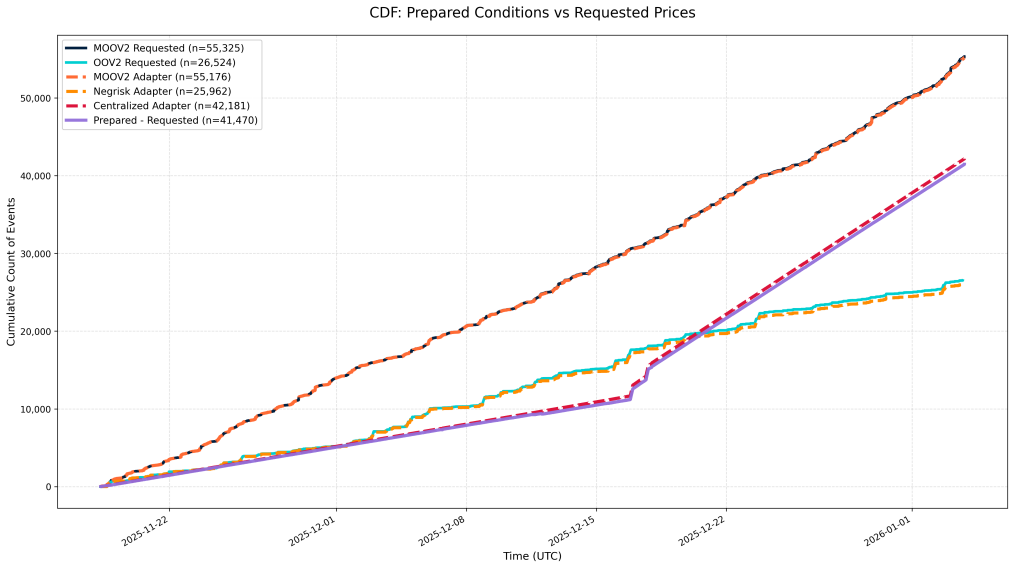

Figure 5.2: UMA Price Requests and Condition Preparation timing

Figure 5.2 is a strong indicator for the fact that the known deployments OOV2 and MOOV2 of the UMA protocol, are almost if not completely exlcusively used by the Negrisk Adapter and MOOV2 adapter respectively. However, we can do better. In section 4.2 it was stated that Polymarket seems to adhere to the UMIP-107 standard for making price requests. This would imply that every identifier should be *YES_OR_NO_QUERY* and every ancillary data should be utf-8 encoded dictionary. Doing a similar analysis as in 3.1 we find that virtually all of the 225814 events emitted in the time range 2025-11-18 07:00:17 to 2026-01-03 19:48:45 from the OOV2 and MOOV2 do in fact adhere to this standard and even extend with the keys: *initializer*, *q*, *title*, *description*, *market_id*. We can now use the market objects to match with RequestPrice events by market id and then use the condition id to matche these pairs with the ConditionPreparation events.

Table 5.2: UMA 81,849 RequestPrice matching with CLOB market objects

Alias	**All Events (%)**	**Matched Events (%)**
MOO2	55,325 (67.59%)	55,201 (99.78%)
OOV2	26,524 (32.41%)	25,693 (96.87%)

Table 5.3: UMA 81,849 RequestPrice matching with 123,319 Conditions Prepared

UMA	Matched Events	Oracle
MOOV2	55'172	MOOV2 Adapter
OOV2	70	MOOV2 Adapter
OOV2	25'562	Negrisk Adapter

Out of the 55'172 matches between conditions prepared using the MOOV2 Adapter and price requests made to the MOOV2, 55'052 share the same block timestamp. This leaves 120 matched condition preparations and price requests which could not have been executed in the same transaction atomically, and therefor are coordinated through multiple separate transaction. Out of these, 70 price requests are made to the OOV2. Which by the immutable reference to the MOOV2 in the MOOV2 Adapter contract, could not have been achieved through the *initialize* method inside of the MOOV2 Adapter.

On the other hand the Conditions Prepared with the Negrisk Adapter, as Oracle, matched exclusively with Price Requests made to the OOV2, which is the oracle it immutably references for price requests. However, for it only 19,832 out of the 25'562 matched had zero time difference and a matching transaction hash, indicating that for 22.4 percent of the conditions prepared, are not atomically executed in the same transaction with the price request.

5.4 Centralized Adapter

The bytecode for the Centralized Adapter is not recognized by prominent bytecode to source indexers like Polygonscan or Etherscan. Decompiling it with heimdall rs without symbol inference and then using an LLM for symbol inference we get the solidity source in **??** and see a similar pattern as in the other adapter but fully centralized with access control and no UMA resolution.

How valid is the statement/method above?

Table 5.4: Centralized Adapter Events

Label	**Appeared In (%)**
Up or Down	39,197 (92.9%)
Crypto Prices	39,197 (92.9%)
Hide From New	39,197 (92.9%)
Recurring	39,197 (92.9%)
Crypto	39,197 (92.9%)
5M	20,278 (48.1%)
15M	17,824 (42.2%)
Solana	9,806 (23.2%)
XRP	9,801 (23.2%)
Ripple	9,801 (23.2%)
Ethereum	9,800 (23.2%)
Bitcoin	9,790 (23.2%)
4H	1,095 (2.6%)

Every market object has a set of labels assigned by polymarket to categorize the event. Table 5.4 shows the exhaustive distribution of these labels in the markets objects matched with the conditions prepared with the Centralized Adapter as oracle. Clearly being tied to crypto related events. If we look up individual matched markets on polymarkets web application, none of them have the a link to the UMA resolution status, for the UMA webapplication, which for other markets exists.

5.5 Exchange Registration

For matched orders on the CTFExchange to be settled, first the token id of the associated CLOB must be registered as explained in x. This emits the TokenRegistered event twice for the same condition but simply with the order of token id arguments flipped. This also means that while every condition id has two orderbooks they can only either both be registered or neither be registered on the CTFExchange.

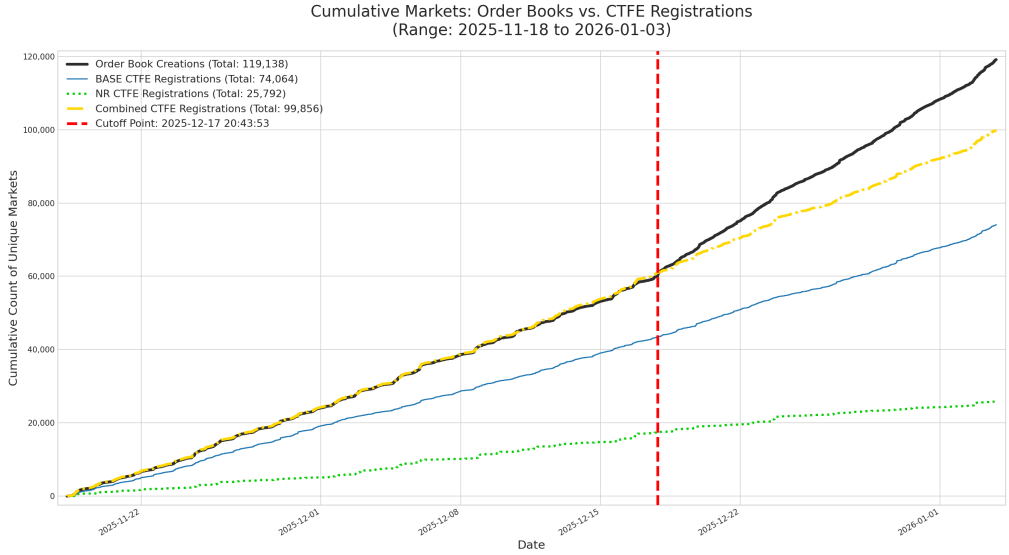

Figure 5.3: CLOB markets and registered tokens unique condito

In figure 5.3 we can see that the market object creations start to diverge from the comined ctfe registritation around the same time that the divergence in 5.1 starts.

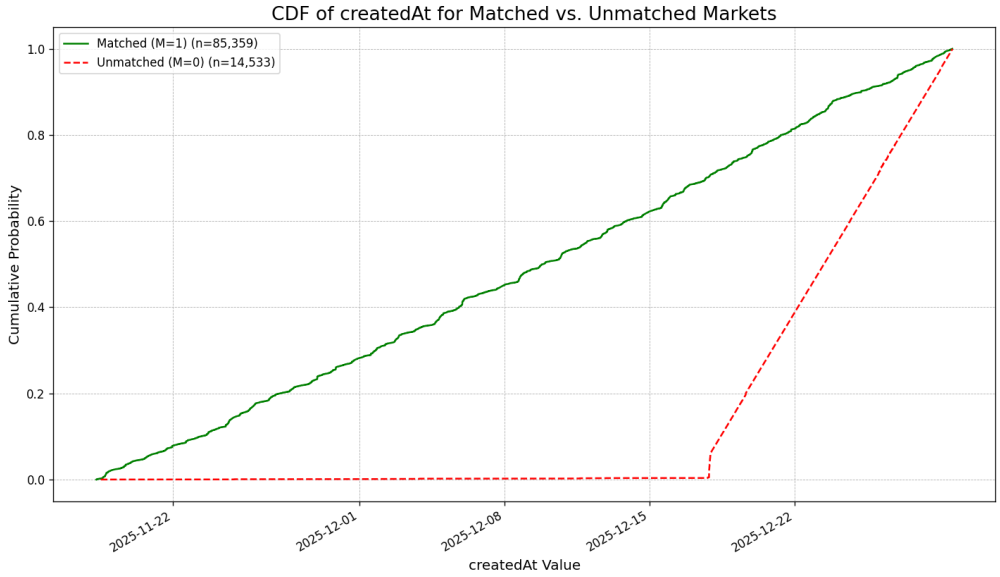

Figure 5.4: CLOB market createAt time and unmatched cdf

In figure 5.4 we can see the actual relationship between the createdAt timestamp and how many unmatched or matched markets exist up until that time.

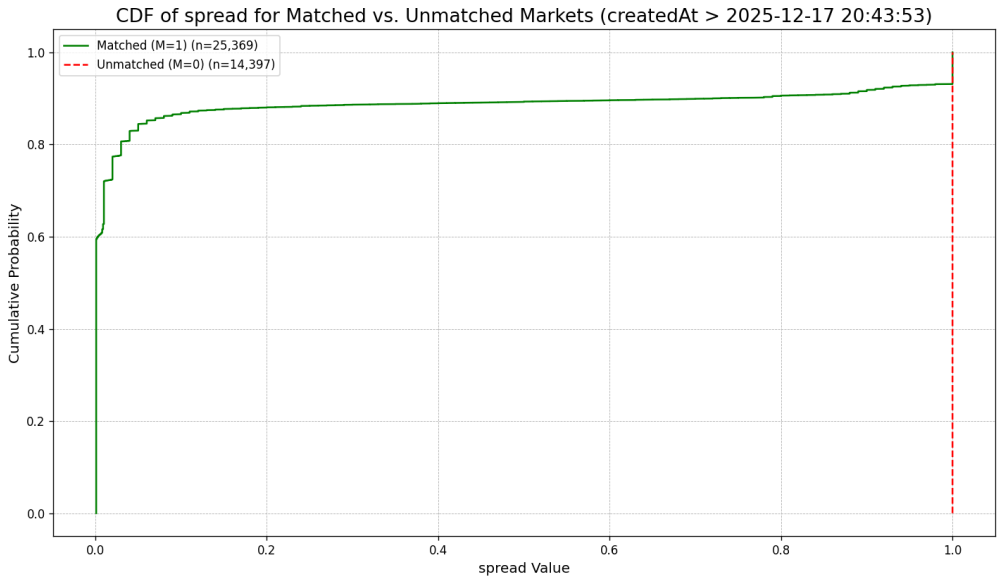

Figure 5.5: CLOB market spread and unmatched cdf

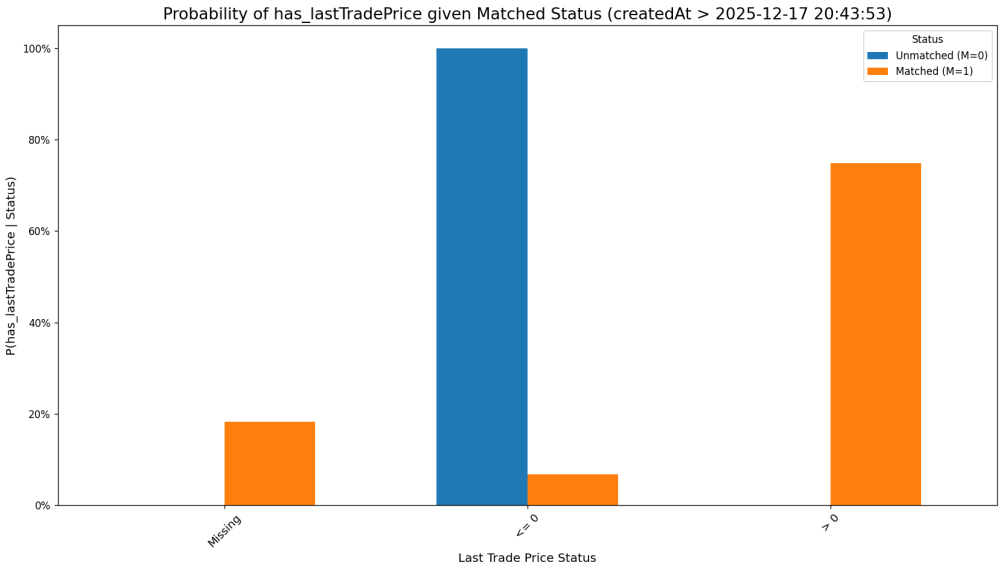

Figure 5.6: CLOB market last trade price and unmatched pmf

In figure 5.4 we clearly see that unmatched markets objects only really start to appear after a specific threshold. The most sensible reason for this would be that the markets have no matched orders yet and therefore don't need to be registerd on the CTFExchange yet. If plot we look at pmf of the three discrete cases for the lastTradePrice attribute from the market object in figure 5.6 after the threshold time, we can clearly see that a missing last trade price or last trade price > 0 implies that the market is matched, whilst for the last trast trades at or below zero we have can only infer an unmatched market with a small probability of false positives. A similar inference with small false positive rate can be done using spread $= 1$, as visible in 5.5.

5.5.1 Unregistered Traded Outcomes

Using the market objects we can match the conditions prepared to the tokens registered events. In table 5.5 we can see that virtually all unregistered markets were prepared using the Centralized Adapter, for which less than half of the conditions are registered. Looking back at table 5.4 we can see many of them are labelled as "Hide From New".

Table 5.5: Matched conditions of 134,424 Conditions Prepared and 215,152 TokensRegistered over Nov 18 - Jan 08

Oracle (Conditions)	**Matches**	**Exchange (Conditions)**
Centralized Adapter (48'896)	20'425	CTFE (83'262)
MOOV2 Adapter (62'857)	61'439	CTFE (83'262)
Negrisk Adapter (27'728)	23'545	Negrisk CTFE (24'209)

However there provably are markets, which are both on the web application and listed as active markets in the CLOB API, with trade and order book state change messages being emitted over the websocket market channel. An example for this, are some of the 15 minute cyclical bitcoin up or down markets, which do not have any registered tokens events nor orders matched or order filled events emitted from neither the CTFE nor Negrisk CTFE.

5.6 Resolution

5.6.1 Disputes

As described in section 4.2 the UMA OOV2 and MOOV2 protocol allows for price proposals to be disputed within a liveness period. If a proposal is disputed a stake based voting process starts, which is off chain coordinated through a message relayer.

Table 5.6: Ten Most Frequent Tags in 462 Matched Disputes

Label	**Appeared In (%)**
Sports	136 (29.44%)
Politics	130 (28.14%)
Games	130 (28.14%)
Culture	123 (26.62%)
Hide From New	72 (15.58%)
Recurring	67 (14.50%)
Weather	61 (13.20%)
Trump	52 (11.26%)
Geopolitics	52 (11.26%)
Best of 2025	41 (8.87%)

There are a total of 462 matched disputes and 113,591 matched propsals. Table 5.6 shows the ten most frequent labels assigned to the markets objects matched with the disputes. Out of the 462 matched disputes, 197 are from the OOV2 and 265 from the MOOV2.

5.6.2 Reported Unkown Outcome

We already saw that there are three oracles polymarket uses to report the payouts to enforce a terminal value for the YES and NO shares after resolution. The terminal value is then determined by the payout array passed to the CFT. In the contracts this is hardcoded you either be [1,0] for YES, [0,1] for NO, or [1,1] for UNKNOWN.

Table 5.7: Ten Most Frequent Tags For Markets resolved to UNKOWN outcome (n=1,011)

Label	**Appeared In (%)**
Games	999 (98.81%)
Sports	986 (97.53%)
Tennis	435 (43.03%)
Esports	403 (39.86%)
counter strike 2	246 (24.33%)
Dota 2	62 (6.13%)
Basketball	57 (5.64%)
UFC	43 (4.25%)
NCAA	43 (4.25%)
NCAA Basketball	31 (3.07%)

Out of 202'514 matched Condition Resolution events only 1'011 had the UN-

KNOWN outcome [1,1] reported as payout array. These 1'011 UNKNOWN outcomes were all reported by the MOOV2 Adapter. Table 5.7 shows the ten most event categories associated with the the markets that were reported to have an UNKNOWN outcome.

5.6.3 Manual Resolutions

In sections **??** it was mentioned taht the UmaCtfAdapter and Negrisk Operator have administrative methods, which can be used to manually resolve markets without UMA price requests being settled. All of the relevant methods emit events, including the *resolveMarketManually* method in the UmaCtfAdapter and the *emergencyResolveQuestion* method in the Negrisk Operator. However, there are no events for either of these methods was emitted in any of the UmaCtfAdapter instances or the single Negrisk Operator instance.

Chapter 6

Orderbook and Trading

Both Polymarket and Kalshi utilize a Central Limit Order Book (CLOB) as their primary market mechanism to match buyers and sellers. Market participants can place orders through the *order* or *orders* endpoint on both Polymarket and Kalshi. An order on both exchanges is specified through a JSON object. The attributes vary by exchange but fundamentally there are two order types:

- **Market orders** guarantee timely execution by attempting to match the quantity specified against existing resting limit orders by **price-time priority**. However, the price for the base assets is not guaranteed.
- **Limit orders** guarantee price for the quantity specified, but not timely execution. As they only get matched if a market order is placed and they have the greatest **price-time priority**.

Every executed order, also simply called trade, consists of a taker, who's order was marketable, matched with at least one maker, who had a resting limit order.

6.1 State, Ordering and Matching

Both Kalshi and Polymarket use **Price-Time Priority** for matching market orders with resting limit orders [**?**]. This means that limit orders at better prices which arrived at the CLOB the soonest, are chosen first to execute against market orders.

On Polymarket there are three ways a maker and taker can get matched as was described in section **??**. The documentation on Polymarket presents a separate orderbook for the YES and NO shares of a prediction market with buy and sell sides. The data collected from the endpoints reveals that there actually only is one orderbook for a prediction market, where for every price p there exists a price $1-p$ on the complementary outcomes complementary side with the exact same amount of shares available at all times. So for example, if there are 50 YES share bids at 0.7 USDC, then there also are 50 NO share asks at 0.3 USDC. For

this reason it can be treated as a single orderbook with YES and NO sides, just like Kalshi presents it.

The state of the CLOB can be defined through the current existing resting limit. A limit order can be defined as a tuple (p, v, s, i), where $p \in \mathbb{R}^+$ is the price for an outcome share, $v \in \mathbb{R}^+$ is the quantity of the shares, $s \in \{\text{YES}, \text{NO}\}$ is the outcome and $i \in \mathbb{N}$ is the priority it has to be selected for matching with market orders at price p and side s, a lower i has a higher priority. This guarantees a unique ordering for both buy and sell limit orders which is used for matching. Formally, for limit orders we have that:

$$((p_1, v_1, s_1, i_1) > (p_2, v_1, s_2, i_2)) \iff p_1 < p_2 \vee (p_1 = p_2 \wedge i_1 < i_2) \tag{6.1}$$

Where the best buy price for the taker is the lowest:

$$p_{market} = max\{p|(p, v, s, i)\} \tag{6.2}$$

The market price p_{market} generically refers to the best price for the taker, regardless of whether its on the YES or NO side. If the quantity for a marketable order can not be fully filled at market price, then the taker would suffer from **price slippage**, if the order is executed. If the marketable order can't be filled with limit orders at any price, then the taker order remains partially or fully unfilled. Depending on the market order specification, a partially unfilled market order is either executed or canceled.

6.2 Slippage

In general, Slippage occurs when the trade is executed at a different price than what the market suggested at the time where the intent to trade was submitted. This was already covered for fully Automated Market Makers in section 4.4.1.

Slippage in a CLOB can also occur when a taker order for quantity x is not filled entirely at the market price when the order was submitted, resulting in a less favorable execution cost for the taker. The CLOB is designed to allow this, with the core principle being the taker is guaranteed timely order execution but not price, whilst the maker is guaranteed order price but not timely execution. We can define the slippage for a taker buying or selling x base assets and getting matched with maker orders M, we have:

$$Slippage(x, M) = |\sum_{(p,v,s,i) \in M} p * v - p_{market} \cdot x| \tag{6.3}$$

The least limit order $(p, v, s, i) = min\{M\}$ might only be part of a limit order such that another limit order (p, v', s, i) now exists where the previous limit order

was (p, v'', s, i) for $v'' = v' + v$ and:

$$x = \sum_{(p,v,s,i) \in M} v \tag{6.4}$$

6.2.1 Which party will win the house 2026?

To compute the slippage for a taker in a trade, we can use the OrderFilled events. These are emitted whenever a trade is settled for every maker order filled by the taker and once for the exchange filling the taker, as explained in section 4.4. However in section 5.5 we also mentioned that not all trades are settled on chain, this also includes some if not all of the 15 minute cyclical bitcoin up or down markets. Since we can not reliably compute slippage from the data available from the endpoints, we can only measure slippage for the "Which party will win the house 2026?" market on Polymarket, which has all of its trades settled on chain.

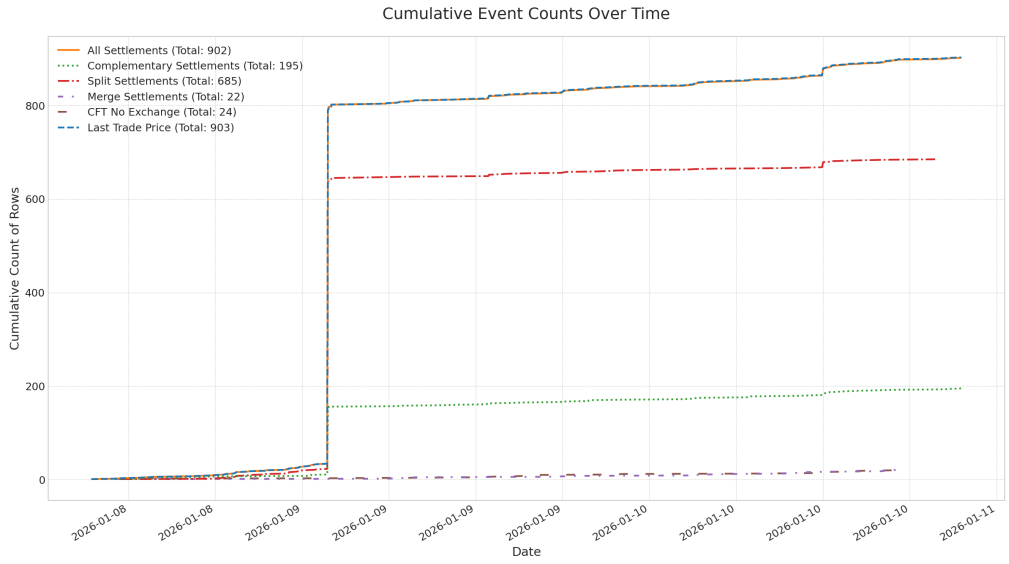

Figure 6.1: Cumulative count of trades for the event "Who will win the house 2026?"

In figure 6.1 we can see that at least for this specific event the settlements reported on chain seem timely with the server timestamp from last trades reported through the CLOB market channel websocket endpoint. Further analysis shows that

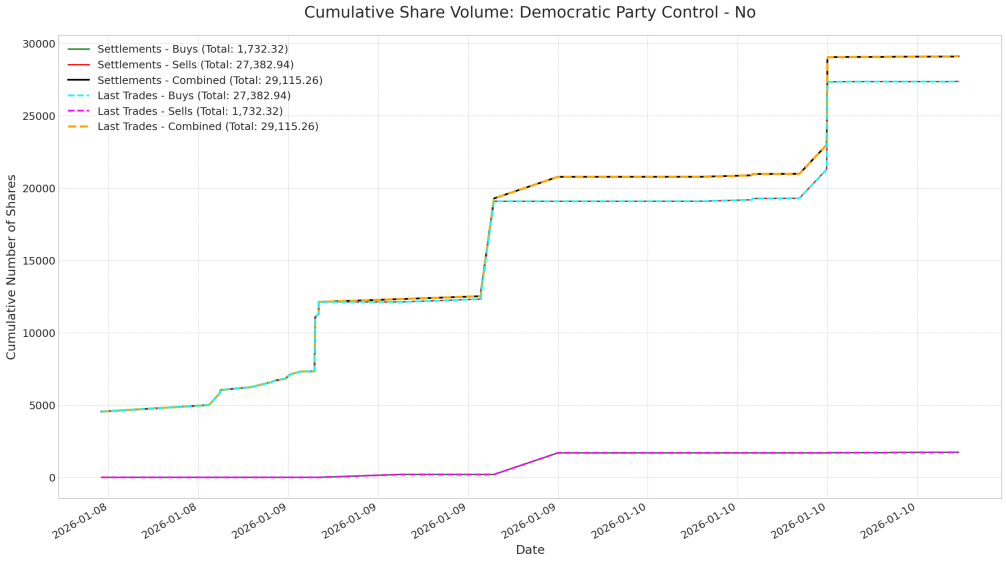

Figure 6.2: Cumulative volume democrats no

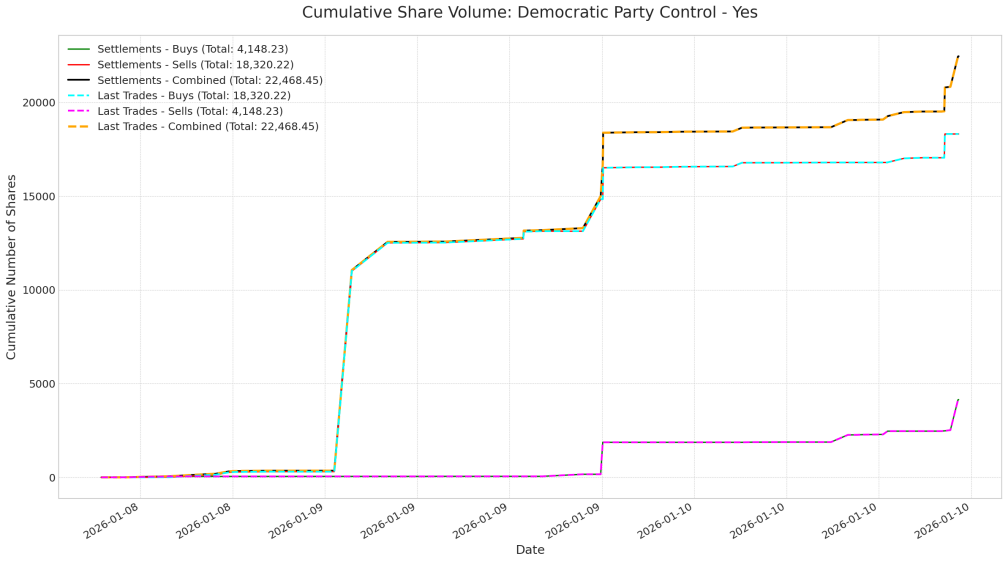

Figure 6.3: Cumulative volume democrats yes

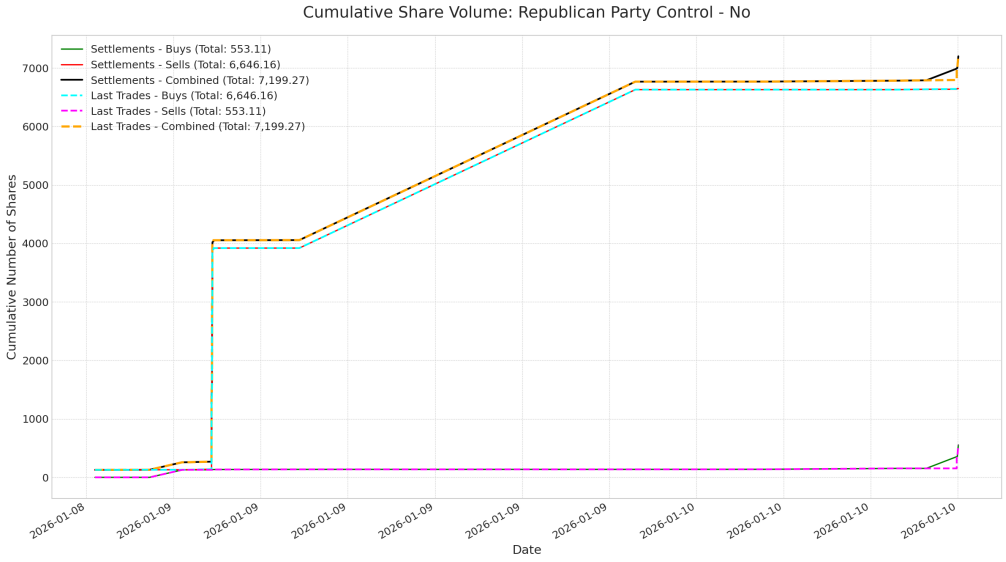

Figure 6.4: Cumulative volume republicans no

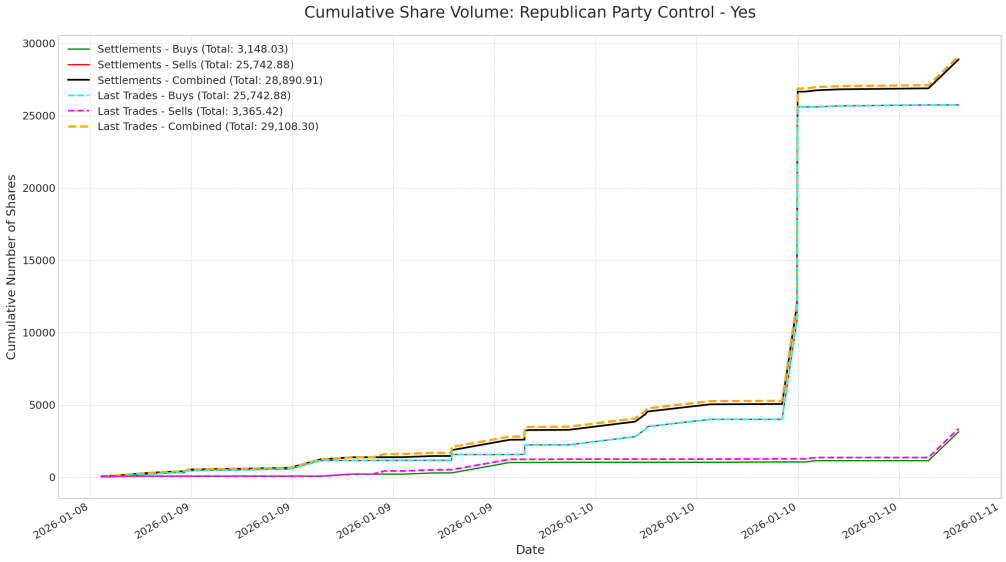

Figure 6.5: Cumulative volume republicans yes

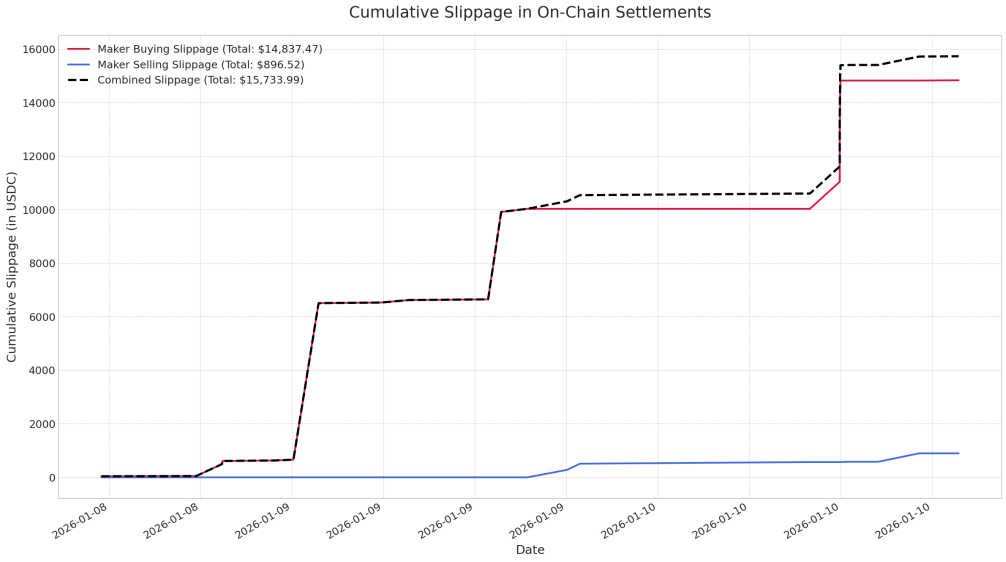

Figure 6.6: Cumulative slippage

In figure 6.6 we can see the cumulative slippage for all trades executed on the "Which party will control the House after the 2026 Midterm elections?" market on Polymarket. This includes both markets for Democrats and Republicans winning the house, and is measured as defined in the relationship 6.3, using the OrderFilled events emitted from the CTFE contract as explained before.

6.3 Arbitrage

The nature of prediction markets nturally induces multiple sources for arbitrage. These all arise from price discrepancies between price pairs p_1, p_2 which depend on each other.

6.3.1 Which party will win the house 2026?

For the event "Which party will control the House after the 2026 Midterm elections?" there are two markets, one for Democrats winning the house and one for Republicans winning the house. For which there are 8 prices in total, 4 on polymarket and 4 on kalshi. This leaves us with 28 combinations of price pairs for arbitrage. For analytic purposes we will restrict ourselves to 6 prices, the YES and NO prices for both the republican and democratic party on Polymarket, and the YES and NO price for the democratic party on Kalshi. We can now measure

3 sources of arbitrage specifically: market internal, complementary market, and mirror market arbitrage.

Define each of these formally and illustrate transitive closure

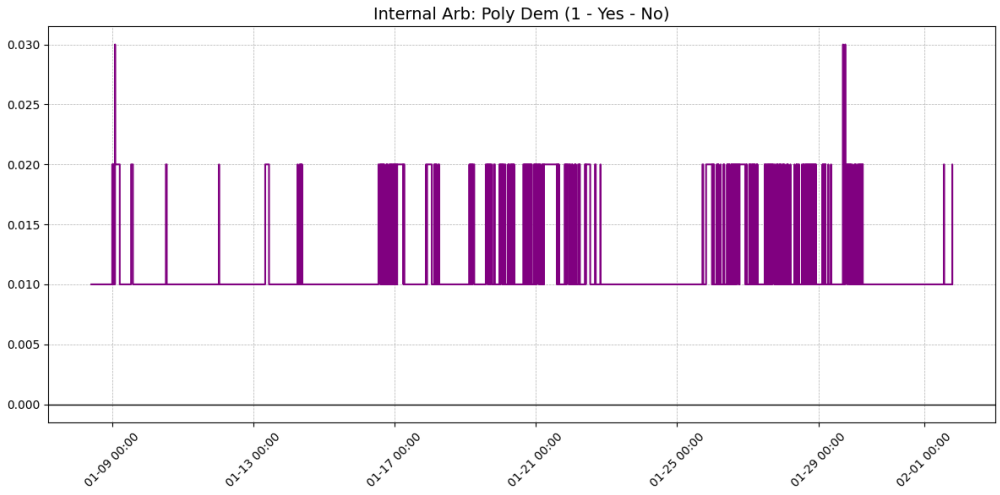

Figure 6.7: Poly Dem Internal

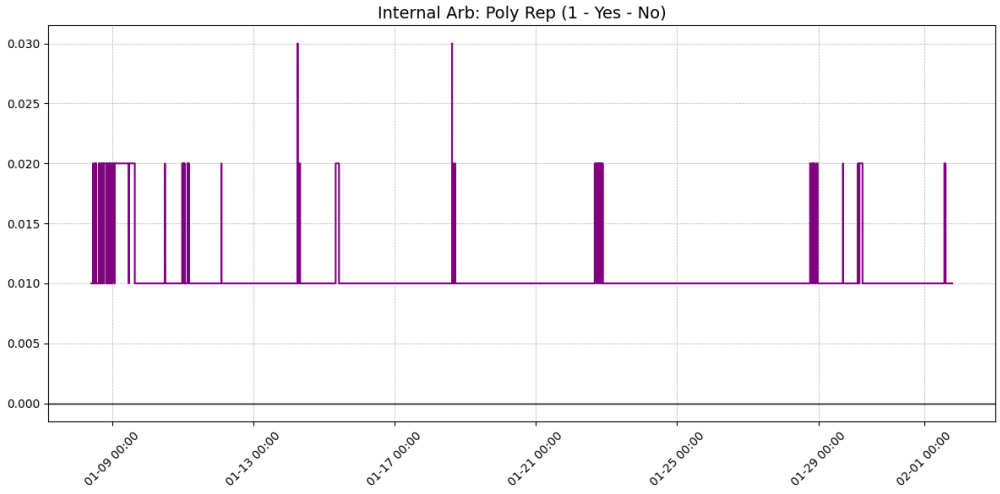

Figure 6.8: Poly Rep Internal

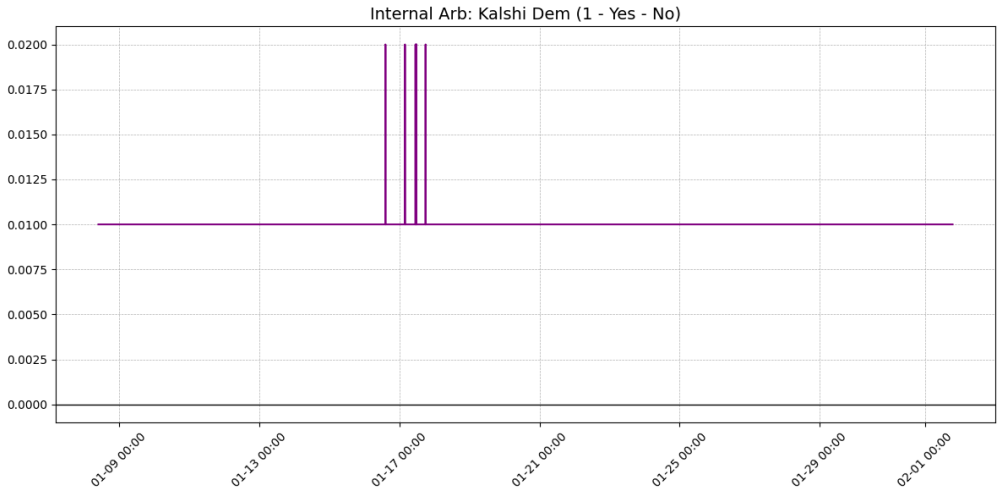

Figure 6.9: Kalshi Dem Internal

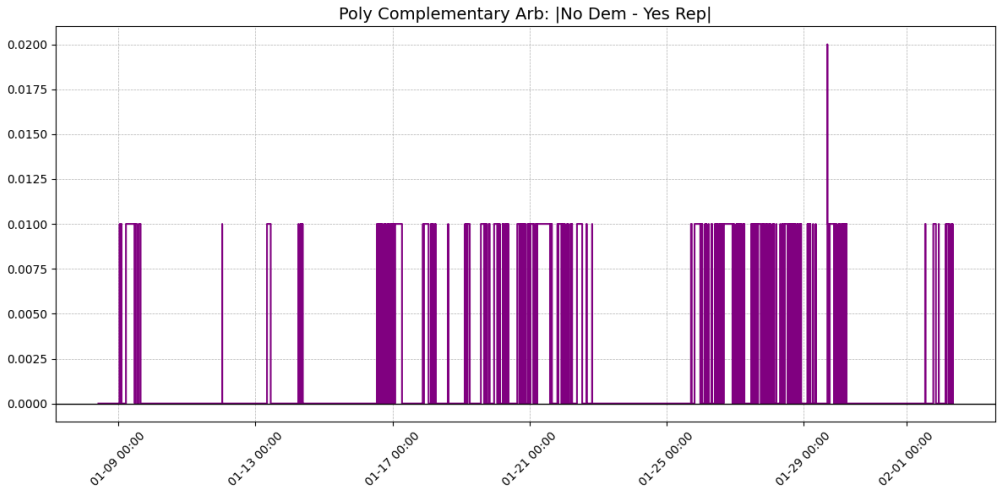

Figure 6.10: Poly Rep Yes and Dem No Complementary

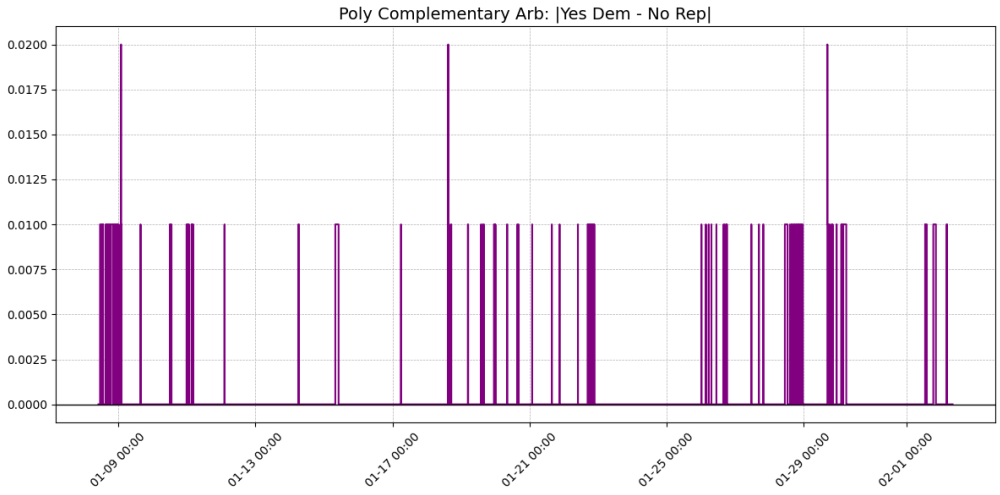

Figure 6.11: Poly Rep No and Dem Yes Complementary

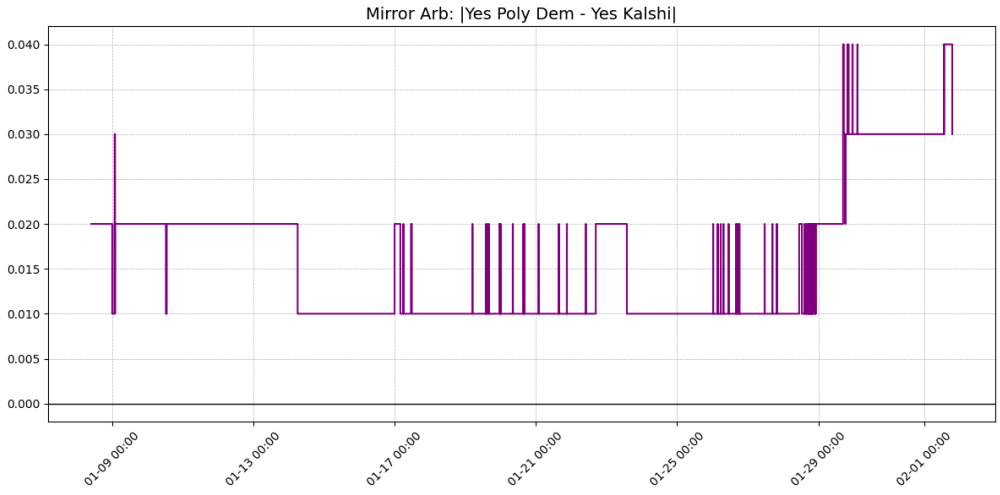

Figure 6.12: Dem Yes Mirror

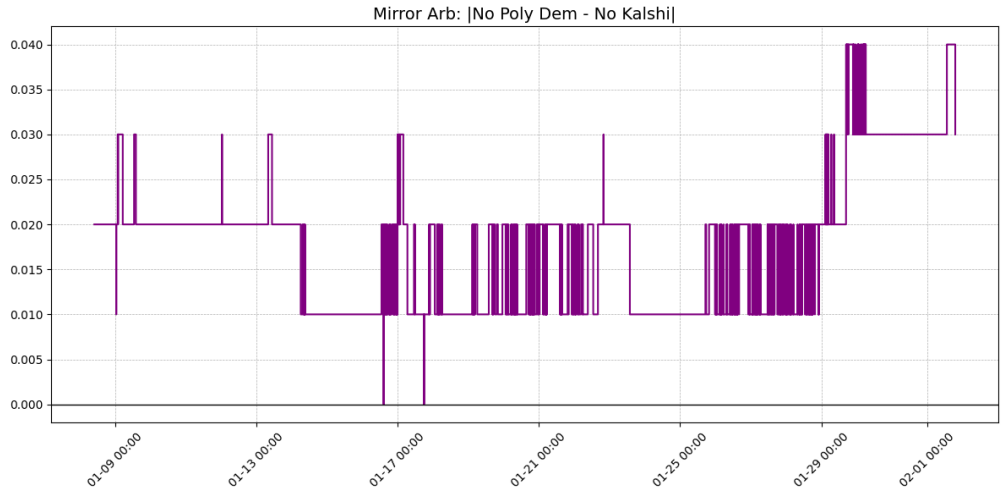

Figure 6.13: Dem No Mirror

Add numerical value the E(price diff), std(price diff) or P(price Difference <= tau) for each arbitrage type for some interesting tau

6.3.2 Bitcoin up or down?

In the prior section we saw that the event "Which party will win the house 2026?" had low expected price discrepancies. However, it is important to also consider that the price for the event itself did not move much, and also was not close to being resolved yet. With the cyclical 15 minute "Bitcoin up or down?" event, we can also see how price discrepancies relate to the timeline in the market life cycle.

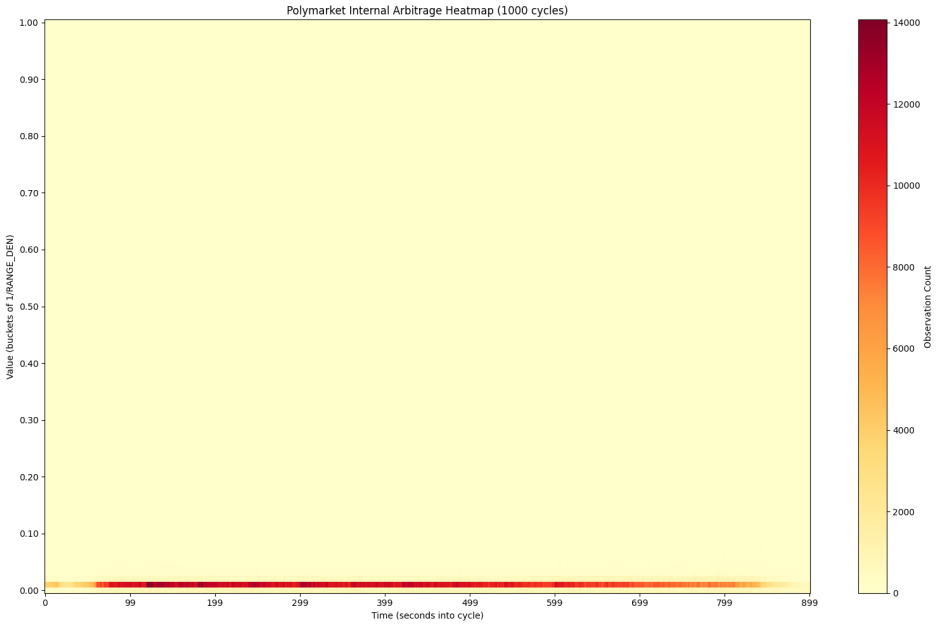

Figure 6.14: Poly Internal

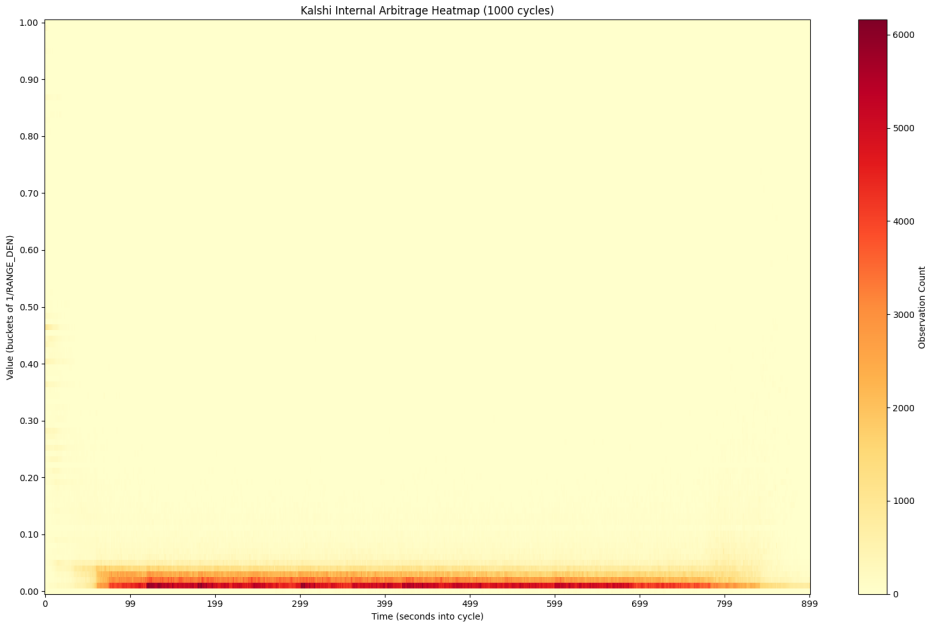

Figure 6.15: Kalshi Internal

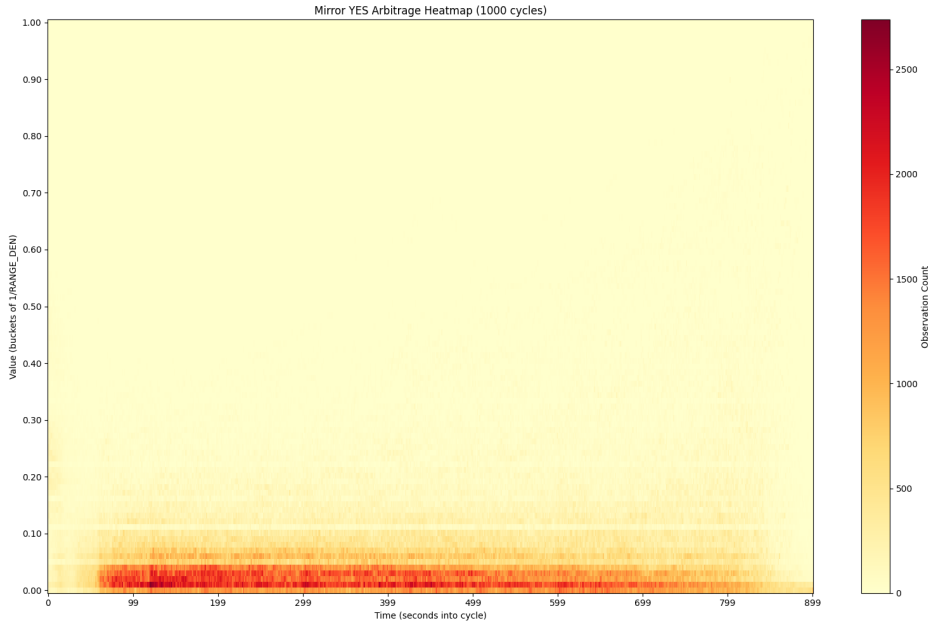

Figure 6.16: Yes Mirror

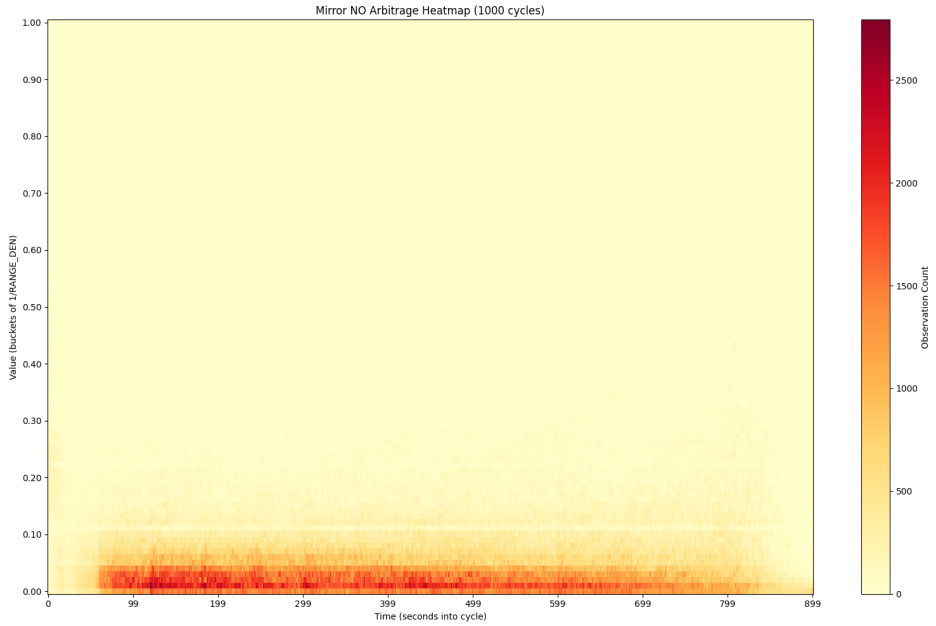

Figure 6.17: No Mirror

The tables above create a heat map using the 15 minute cyclical "Bitcoin up or down?" events, to visualize the distribution of price discrepancies at a given time in the life cycle of the market.

better illustrate how cross exchange (mirror arb) are more variant with higher expected value. Also how in general variance decreases at start and increases at the end of the market life cycle.

Chapter 7

Discussion

Love

Bibliography

[1] L. A. Stout, "Why the law hates speculators: Regulation and private ordering in the market for otc derivatives," *Duke Law Journal*, vol. 48, p. 701, 1999.

[2] U.S. Supreme Court, "Board of trade of city of chicago v. christie grain & stock co., 198 u.s. 236," 1905.

[3] E. Lambert, *The Onion Futures Act*. Basic Books, 2010, ch. The Onion Futures Act.

[4] United States Congress, "Onion futures act," Pub. L. 85–839, 72 Stat. 1013, Aug. 1958, codified at 7 U.S.C. § 13-1. [Online]. Available: https://www.law.cornell.edu/uscode/text/7/13-1

[5] Code of Federal Regulations, "Review of event contracts based upon certain excluded commodities," 17 C.F.R. § 40.11, 2021, prohibition on contracts involving terrorism, assassination, war, and gaming.

[6] U.S. Commodity Futures Trading Commission, "In re Blockratize, Inc. d/b/a Polymarket.com," CFTC Docket No. 22-09, January 2022, order Instituting Proceedings Pursuant to Section 6(c) and 6(d) of the Commodity Exchange Act.

[7] "KalshiEx LLC v. Commodity Futures Trading Commission," D.D.C., September 2024, memorandum Opinion granting summary judgment to Plaintiff.

[8] Kalshi, "Kalshi member agreement," 2026. [Online]. Available: https://kalshi.com/docs/kalshi-member-agreement.pdf

[9] Token Terminal, "Kalshi metrics: Trading volume," Token Terminal, 2025, data covering the year 2025. Accessed January 2, 2026. [Online]. Available: https://tokenterminal.com/explorer/projects/kalshi/metrics/trading-volume?interval=365d

[10] M. Cattaneo, "Dune analytics query for polymarket trading volume," Dune, 2025, query ID: 6401125. Data covering the year 2025. Accessed January 2, 2026. [Online]. Available: https://dune.com/queries/6401125

[11] P. Graham, "Startup = growth," PaulGraham.com, September 2012. [Online]. Available: http://www.paulgraham.com/growth.html

[12] J. Clinton and T.-F. Huang, "Prediction markets? the accuracy and efficiency of $2.4 billion in the 2024 presidential election," OSF Preprints, December 2025. [Online]. Available: https://osf.io/preprints/socarxiv/a3v5p

[13] J. Wolfers and J. E. Stiglitz, "Prediction markets: The lessons from the 2024 election," NBER Working Paper, December 2024, working Paper w33005. [Online]. Available: https://www.nber.org/papers/w33005

[14] G. Gottsegen, "In 2024, prediction markets called the presidential election before the polls could. now, they're mostly betting on sports," Morningstar, December 2024. [Online]. Available: https://www.morningstar.com/news/marketwatch/20241220282/in-2024-prediction-markets-called-the-presidential-election-before-the-polls-could-now-theyre-m

[15] N. De, "Polymarket's main 2024 u.s. presidential election market sees $3.5b in volume," CoinDesk, November 2024. [Online]. Available: https://www.coindesk.com/policy/2024/11/06/polymarkets-main-2024-us-presidential-election-market-sees-35b-in-volume/

[16] A. Sirolly, H. Ma, Y. Kanoria, and R. Sethi, "Network-based detection of wash trading (november 06, 2025)," SSRN, November 2025. [Online]. Available: https://ssrn.com/abstract=5714122orhttp://dx.doi.org/10.2139/ssrn.5714122

[17] M. Cattaneo, "Polymarket's trading volume in november and december of 2025," Dune, January 2026. [Online]. Available: https://dune.com/queries/6452777?utm_source=share&utm_medium=copy&utm_campaign=query

[18] Kalshi, "Kalshi is now available in 140+ countries," Kalshi Blog, October 2025. [Online]. Available: https://kalshi.com/blog/kalshi-is-now-available-in-140-countries

[19] Markets Wiki. Qcx llc. [Online]. Available: https://www.marketswiki.com/wiki/QCX,_LLC

[20] CFTC. Qcx llc dcm exchange registry. [Online]. Available: https://www.cftc.gov/IndustryOversight/IndustryFilings/TradingOrganizations/49571

[21] Swiss Gambling Supervisory Authority (GESPA), "Liste der gesperrten zugänge (blocklist)," Official Publication, October 2025, accessed January 2, 2026. The specific PDF is dated October 21, 2025. [Online]. Available: https://www.gespa.ch/de/bekaempfung-illegaler-aktivitaeten/zugangssperre

[22] D. Stabile, "Kalshi sues illinois regulators over cease-and-desist order," Gaming Today, November 2025. [Online]. Available: https://www.gamingtoday.com/news/kalshi-sues-illinois-regulators-over-cease-and-desist-order/

[23] Polymarket, "Polymarket contract security," Github, November 2025. [Online]. Available: https://github.com/Polymarket/contract-security

[24] UMAprotocol, "Umip-189," Github, November 2025. [Online]. Available: https://github.com/UMAprotocol/UMIPs/blob/master/UMIPs/umip-189.md

[25] UMAprotocol, "Umip-107," Github, November 2025. [Online]. Available: https://github.com/UMAprotocol/UMIPs/blob/master/UMIPs/umip-107.md

[26] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 910–927.

[27] L. Heimbach and R. Wattenhofer, "Eliminating sandwich attacks with the help of game theory," in *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. ACM, 2022, pp. 612–626.

[28] Kalshi, "Kalshi tokenized contracts," December 2025. [Online]. Available: https://news.kalshi.com/p/kalshi-solana-tokenized-predictions

[29] O. Saguillo, V. Ghafouri, L. Kiffer, and G. Suarez-Tangil, "Unravelling the probabilistic forest: Arbitrage in prediction markets," 2025. [Online]. Available: https://arxiv.org/abs/2508.03474

Appendix A

Endpoint Documented JSON responses

Listing A.1: Polymarkets Documented Orderbook State Object Sample

```
{
  "market": "0x1b6f76e5b8587ee896c35847e12d
  11e75290a8c3934c5952e8a9d6e4c6f03cfa",
  "asset_id": "1234567890",
  "timestamp": "2023-10-01T12:00:00Z",
  "hash": "0xabc123def456...",
  "bids": [
    {
      "price": "1800.50",
      "size": "10.5"
    }
  ],
  "asks": [
    {
      "price": "1800.50",
      "size": "10.5"
    }
  ],
  "min_order_size": "0.001",
  "tick_size": "0.01",
  "neg_risk": false
}
```

Listing A.2: Polymarkets Documented Book Event

```
{
  "event_type": "book",
  "asset_id": "65818619657568813474341868652308942079804919
  2873804221928922111314087931254 22",
  "market": "0xbd31dc8a20211944f6b70f31557f1001557
  b59905b7738480ca09bd4532f84af",
  "bids": [
    { "price": ".48", "size": "30" },
    { "price": ".49", "size": "20" },
    { "price": ".50", "size": "15" }
```

```
  ],
  "asks": [
    { "price": ".52", "size": "25" },
    { "price": ".53", "size": "60" },
    { "price": ".54", "size": "10" }
  ],
  "timestamp": "123456789000",
  "hash": "0x0...."
}
```

Listing A.3: Polymarkets Documented Price Change Event

```
{
    "market": "0x5f65177b394277fd294cd75650044e32
    ba009a95022d88a0c1d565897d72f8f1",
    "price_changes": [
        {
            "asset_id": "71321045679252212594626385532706912
            75033272857194253228963137931245558399256 3",
            "price": "0.5",
            "size": "200",
            "side": "BUY",
            "hash": "56621a121a47ed9333273e21c83b660cff37ae50",
            "best_bid": "0.5",
            "best_ask": "1"
        },
        {
            "asset_id": "52114319501245915516055106046884 2
            09969926127482827954674443846427813813222426",
            "price": "0.5",
            "size": "200",
            "side": "SELL",
            "hash": "1895759e4df7a796bf4f1c5a5950b748306923e2",
            "best_bid": "0",
            "best_ask": "0.5"
        }
    ],
    "timestamp": "1757908892351",
    "event_type": "price_change"
}
```

Listing A.4: Polymarkets Documented Last Trade Price Event

```
{
"asset_id":"114122071509644379678018727908709560
226618148003371446110114509806601493071694",
"event_type":"last_trade_price",
"fee_rate_bps":"0",
"market":"0x6a67b9d828d53862160e470329ffea5246f3
38ecfffdf2cab45211ec578b0347",
"price":"0.456",
"side":"BUY",
"size":"219.217767",
"timestamp":"1750428146322"
```

```
}
```

Listing A.5: Polymarkets Documented New Market Event

```
{
    "id": "1031769",
    "question": "Will NVIDIA (NVDA) close above
    $240 end of January?",
    "market": "0x311d0c4b6671ab54af4970c06fcf
    58662516f5168997bdda209ec3db5aa6b0c1",
    "slug": "nvda-above-240-on-january-30-2026",
    "description": "This market will resolve to \"Yes\" if the
        official closing price for NVIDIA (NVDA) on the final
        trading day of January 2026 is higher than the listed price
        . Otherwise, this market will resolve to \"No\".\n\nIf the
        final trading day of the month is shortened (for example,
        due to a market-holiday schedule), the official closing
        price published for that shortened session will still be
        used for resolution.\n\nIf no official closing price is
        published for that session (for example, due to a trading
        halt into the close, system issue, or other disruption),
        the market will use the last valid on-exchange trade price
        of the regular session as the effective closing price.\n\
        nThe resolution source for this market is Yahoo Finance
        specifically, the NVIDIA (NVDA) \"Close\" prices available
        at https://finance.yahoo.com/quote/NVDA/history, published
        under \"Historical Prices.\"\n\nIn the event of a stock
        split, reverse stock split, or similar corporate action
        affecting the listed company during the listed time frame,
        this market will resolve based on split-adjusted prices as
        displayed on Yahoo Finance.",
    "assets_ids": [
        "76043073756653678226373981964075571318
        26728924813471736928451899592278932642 5",
        "31690934263385727664202099278545688007 7
        99199447969475608906331829650099442770"
    ],
    "outcomes": [
        "Yes",
        "No"
    ],
    "event_message": {
        "id": "125819",
        "ticker": "nvda-above-in-january-2026",
        "slug": "nvda-above-in-january-2026",
        "title": "Will NVIDIA (NVDA) close above ___ end of January
            ?",
        "description": "This market will resolve to \"Yes\" if the
            official closing price for NVIDIA (NVDA) on the final
            trading day of January 2026 is higher than the listed
            price. Otherwise, this market will resolve to \"No\".\n
            \nIf the final trading day of the month is shortened (
            for example, due to a market-holiday schedule), the
            official closing price published for that shortened
```

```
            session will still be used for resolution.\n\nIf no
            official closing price is published for that session (
            for example, due to a trading halt into the close,
            system issue, or other disruption), the market will use
             the last valid on-exchange trade price of the regular
            session as the effective closing price.\n\nThe
            resolution source for this market is Yahoo Finance
            specifically, the NVIDIA (NVDA) \"Close\" prices
            available at https://finance.yahoo.com/quote/NVDA/
            history, published under \"Historical Prices.\"\n\nIn
            the event of a stock split, reverse stock split, or
            similar corporate action affecting the listed company
            during the listed time frame, this market will resolve
            based on split-adjusted prices as displayed on Yahoo
            Finance."
    },
    "timestamp": "1766790415550",
    "event_type": "new_market"
}
```

Listing A.6: Polymarkets Documented Market Resolved Event

```
{
    "id": "1031769",
    "question": "Will NVIDIA (NVDA) close above $240
    end of January?",
    "market": "0x311d0c4b6671ab54af4970c06fc
    f58662516f5168997bdda209ec3db5aa6b0c1",
    "slug": "nvda-above-240-on-january-30-2026",
    "description": "This market will resolve to \"Yes\" if the
        official closing price for NVIDIA (NVDA) on the final
        trading day of January 2026 is higher than the listed price
        . Otherwise, this market will resolve to \"No\".\n\nIf the
        final trading day of the month is shortened (for example,
        due to a market-holiday schedule), the official closing
        price published for that shortened session will still be
        used for resolution.\n\nIf no official closing price is
        published for that session (for example, due to a trading
        halt into the close, system issue, or other disruption),
        the market will use the last valid on-exchange trade price
        of the regular session as the effective closing price.\n\
        nThe resolution source for this market is Yahoo Finance
        specifically, the NVIDIA (NVDA) \"Close\" prices available
        at https://finance.yahoo.com/quote/NVDA/history, published
        under \"Historical Prices.\"\n\nIn the event of a stock
        split, reverse stock split, or similar corporate action
        affecting the listed company during the listed time frame,
        this market will resolve based on split-adjusted prices as
        displayed on Yahoo Finance.",
    "assets_ids": [
        "76043073756653678226373981964075571318 2
        67289248134717369284518995922789326425",
        "31690934263385727664202099278545688007 7
        99199447969475608906331829650099442770"
```

```
    ],
    "winning_asset_id": "76043073756653678226373981964075571318267289248134717369284518995922789326425",
    "winning_outcome": "Yes",
    "event_message": {
        "id": "125819",
        "ticker": "nvda-above-in-january-2026",
        "slug": "nvda-above-in-january-2026",
        "title": "Will NVIDIA (NVDA) close above ___ end of January?",
        "description": "This market will resolve to \"Yes\" if the official closing price for NVIDIA (NVDA) on the final trading day of January 2026 is higher than the listed price. Otherwise, this market will resolve to \"No\".\n\nIf the final trading day of the month is shortened (for example, due to a market-holiday schedule), the official closing price published for that shortened session will still be used for resolution.\n\nIf no official closing price is published for that session (for example, due to a trading halt into the close, system issue, or other disruption), the market will use the last valid on-exchange trade price of the regular session as the effective closing price.\n\nThe resolution source for this market is Yahoo Finance specifically, the NVIDIA (NVDA) \"Close\" prices available at https://finance.yahoo.com/quote/NVDA/history, published under \"Historical Prices.\"\n\nIn the event of a stock split, reverse stock split, or similar corporate action affecting the listed company during the listed time frame, this market will resolve based on split-adjusted prices as displayed on Yahoo Finance."
    },
    "timestamp": "1766790415550",
    "event_type": "new_market"
}
```


Listing A.7: Kalshi Documented Orderbook Delta

Listing A.8: Kalshi Documented Orderbook Snapshot


```
{
  "type": "orderbook_delta",
  "sid": 2,
  "seq": 3,
  "msg": {
    "market_ticker": "FED-23DEC-T3.00",
    "market_id": "9b0f6b43-5b68-4f9f-9f02-9a2d1b8ac1a1",
    "price": 96,
    "price_dollars": "0.960",
    "delta": -54,
    "delta_fp": "-54.00",
    "side": "yes",
```

```
    "ts": "2022-11-22T20:44:01Z"
  }
}
```

Listing A.9: Kalshi Documented Orderbook Snapshot

```
{
  "type": "orderbook_snapshot",
  "sid": 2,
  "seq": 2,
  "msg": {
    "market_ticker": "FED-23DEC-T3.00",
    "market_id": "9b0f6b43-5b68-4f9f-9f02-9a2d1b8ac1a1",
    "yes": [
      [
        8,
        300
      ],
      [
        22,
        333
      ]
    ],
    "yes_dollars": [
      [
        "0.080",
        300
      ],
      [
        "0.220",
        333
      ]
    ],
    "yes_dollars_fp": [
      [
        "0.0800",
        "300.00"
      ],
      [
        "0.2200",
        "333.00"
      ]
    ],
    "no": [
      [
        54,
        20
      ],
      [
        56,
        146
      ]
    ],
    "no_dollars": [
```

```
      [
        "0.540",
        20
      ],
      [
        "0.560",
        146
      ]
    ],
    "no_dollars_fp": [
      [
        "0.5400",
        "20.00"
      ],
      [
        "0.5600",
        "146.00"
      ]
    ]
  }
}
```

Listing A.10: Kalshi Documented Trade Update

```
{
  "type": "trade",
  "sid": 11,
  "msg": {
    "trade_id": "d91bc706-ee49-470d-82d8-11418bda6fed",
    "market_ticker": "HIGHNY-22DEC23-B53.5",
    "yes_price": 36,
    "yes_price_dollars": "0.360",
    "no_price": 64,
    "no_price_dollars": "0.640",
    "count": 136,
    "count_fp": "136.00",
    "taker_side": "no",
    "ts": 1669149841
  }
}
```

Listing A.11: Kalshi Documented Market Lifecycle V2

```
{
  "type": "market_lifecycle_v2",
  "sid": 13,
  "msg": {
    "market_ticker": "INXD-23SEP14-B4487",
    "event_type": "created",
    "open_ts": 1694635200,
    "close_ts": 1694721600,
    "additional_metadata": {
      "name": "S&P 500 daily return on Sep 14",
      "title": "S&P 500 closes up by 0.02% or more",
```

```
      "yes_sub_title": "S&P 500 closes up 0.02%+",
      "no_sub_title": "S&P 500 closes up <0.02%",
      "rules_primary": "The S&P 500 index level at 4:00 PM ET...",
      "rules_secondary": "",
      "can_close_early": true,
      "event_ticker": "INXD-23SEP14",
      "expected_expiration_ts": 1694721600,
      "strike_type": "greater",
      "floor_strike": 4487
    }
  }
}
```

Listing A.12: Kalshi Documented Event Object

```
{
  "event": {
    "event_ticker": "<string>",
    "series_ticker": "<string>",
    "sub_title": "<string>",
    "title": "<string>",
    "collateral_return_type": "<string>",
    "mutually_exclusive": true,
    "category": "<string>",
    "available_on_brokers": true,
    "product_metadata": {},
    "strike_date": "2023-11-07T05:31:56Z",
    "strike_period": "<string>",
    "markets": [
      {
        "ticker": "<string>",
        "event_ticker": "<string>",
        "market_type": "binary",
        "title": "<string>",
        "subtitle": "<string>",
        "yes_sub_title": "<string>",
        "no_sub_title": "<string>",
        "created_time": "2023-11-07T05:31:56Z",
        "updated_time": "2023-11-07T05:31:56Z",
        "open_time": "2023-11-07T05:31:56Z",
        "close_time": "2023-11-07T05:31:56Z",
        "expiration_time": "2023-11-07T05:31:56Z",
        "latest_expiration_time": "2023-11-07T05:31:56Z",
        "settlement_timer_seconds": 123,
        "status": "initialized",
        "response_price_units": "usd_cent",
        "yes_bid": 123,
        "yes_bid_dollars": "0.5600",
        "yes_ask": 123,
        "yes_ask_dollars": "0.5600",
        "no_bid": 123,
        "no_bid_dollars": "0.5600",
        "no_ask": 123,
        "no_ask_dollars": "0.5600",
```

```
        "last_price": 123,
        "last_price_dollars": "0.5600",
        "volume": 123,
        "volume_fp": "10.00",
        "volume_24h": 123,
        "volume_24h_fp": "10.00",
        "result": "yes",
        "can_close_early": true,
        "open_interest": 123,
        "open_interest_fp": "10.00",
        "notional_value": 123,
        "notional_value_dollars": "0.5600",
        "previous_yes_bid": 123,
        "previous_yes_bid_dollars": "0.5600",
        "previous_yes_ask": 123,
        "previous_yes_ask_dollars": "0.5600",
        "previous_price": 123,
        "previous_price_dollars": "0.5600",
        "liquidity": 123,
        "liquidity_dollars": "0.5600",
        "expiration_value": "<string>",
        "tick_size": 123,
        "rules_primary": "<string>",
        "rules_secondary": "<string>",
        "price_level_structure": "<string>",
        "price_ranges": [
          {
            "start": "<string>",
            "end": "<string>",
            "step": "<string>"
          }
        ],
        "expected_expiration_time": "2023-11-07T05:31:56Z",
        "settlement_value": 123,
        "settlement_value_dollars": "0.5600",
        "settlement_ts": "2023-11-07T05:31:56Z",
        "fee_waiver_expiration_time": "2023-11-07T05:31:56Z",
        "early_close_condition": "<string>",
        "strike_type": "greater",
        "floor_strike": 123,
        "cap_strike": 123,
        "functional_strike": "<string>",
        "custom_strike": {},
        "mve_collection_ticker": "<string>",
        "mve_selected_legs": [
          {
            "event_ticker": "<string>",
            "market_ticker": "<string>",
            "side": "<string>",
            "yes_settlement_value_dollars": "0.5600"
          }
        ],
        "primary_participant_key": "<string>",
        "is_provisional": true
```

```
      }
    ]
  },
  "markets": [
    {
      "ticker": "<string>",
      "event_ticker": "<string>",
      "market_type": "binary",
      "title": "<string>",
      "subtitle": "<string>",
      "yes_sub_title": "<string>",
      "no_sub_title": "<string>",
      "created_time": "2023-11-07T05:31:56Z",
      "updated_time": "2023-11-07T05:31:56Z",
      "open_time": "2023-11-07T05:31:56Z",
      "close_time": "2023-11-07T05:31:56Z",
      "expiration_time": "2023-11-07T05:31:56Z",
      "latest_expiration_time": "2023-11-07T05:31:56Z",
      "settlement_timer_seconds": 123,
      "status": "initialized",
      "response_price_units": "usd_cent",
      "yes_bid": 123,
      "yes_bid_dollars": "0.5600",
      "yes_ask": 123,
      "yes_ask_dollars": "0.5600",
      "no_bid": 123,
      "no_bid_dollars": "0.5600",
      "no_ask": 123,
      "no_ask_dollars": "0.5600",
      "last_price": 123,
      "last_price_dollars": "0.5600",
      "volume": 123,
      "volume_fp": "10.00",
      "volume_24h": 123,
      "volume_24h_fp": "10.00",
      "result": "yes",
      "can_close_early": true,
      "open_interest": 123,
      "open_interest_fp": "10.00",
      "notional_value": 123,
      "notional_value_dollars": "0.5600",
      "previous_yes_bid": 123,
      "previous_yes_bid_dollars": "0.5600",
      "previous_yes_ask": 123,
      "previous_yes_ask_dollars": "0.5600",
      "previous_price": 123,
      "previous_price_dollars": "0.5600",
      "liquidity": 123,
      "liquidity_dollars": "0.5600",
      "expiration_value": "<string>",
      "tick_size": 123,
      "rules_primary": "<string>",
      "rules_secondary": "<string>",
      "price_level_structure": "<string>",
```

```
      "price_ranges": [
        {
          "start": "<string>",
          "end": "<string>",
          "step": "<string>"
        }
      ],
      "expected_expiration_time": "2023-11-07T05:31:56Z",
      "settlement_value": 123,
      "settlement_value_dollars": "0.5600",
      "settlement_ts": "2023-11-07T05:31:56Z",
      "fee_waiver_expiration_time": "2023-11-07T05:31:56Z",
      "early_close_condition": "<string>",
      "strike_type": "greater",
      "floor_strike": 123,
      "cap_strike": 123,
      "functional_strike": "<string>",
      "custom_strike": {},
      "mve_collection_ticker": "<string>",
      "mve_selected_legs": [
        {
          "event_ticker": "<string>",
          "market_ticker": "<string>",
          "side": "<string>",
          "yes_settlement_value_dollars": "0.5600"
        }
      ],
      "primary_participant_key": "<string>",
      "is_provisional": true
    }
  ]
}
```

Appendix B

Smart Contracts

Table B.1: Contract Aliases, Deployment Addresses, and Event Tables

Alias	**Deployment Address**	**Contract Table Name**
Base CTFExchange	0x4bfb41d5b3570defd03c39a9a4d8de6bd8b8982e	CTFExchange
Negrisk CTFExchange	0xC5d563A36AE78145C45a50134d48A1215220f80a	CTFExchange
CTF	0x4d97dcd97ec945f40cf65f87097ace5ea0476045	(Gnosis) Conditional Tokens
Negrisk Operator	0x71523d0f655b41e805cec45b17163f528b59b820	Negrisk Operator
Negrisk Adapter	0xd91e80cf2e7be2e162c6513ced06f1dd0da35296	Negrisk Adapter
Negrisk Adapter	0x2F5e3684cb1F318ec51b00Edba38d79Ac2c0aA9d	UmaCtfAdapter
Base UmaCtfAdapter	0x6A9D222616C90FcA5754cd1333cFD9b7fb6a4F74	UmaCtfAdapter
MOOV2 Adapter	0x65070BE91477460D8A7AeEb94ef92fe056C2f2A7	UmaCtfAdapter
MOOV2	0x2C0367a9DB231dDeBd88a94b4f6461a6e47C58B1	(UMA) Managed Optimistic Oracle V2
Oracle Child Tunnel	0xac60353a54873c446101216829a6A98cDbbC3f3D	Oracle Child Tunnel
FxTunnel	0x8397259c983751DAf40400790063935a11afa28a	FxTunnel
OOV2	0xeE3Afe347D5C74317041E2618C49534dAf887c24	(UMA) Optimistic Oracle V2

Table B.2: Contract Event Signatures

Contract / Event Signatures
CTFExchange

Table B.2 – continued from previous page

Contract / Event Signatures
`FeeCharged(receiver(address), tokenId(uint256), fee(uint256))`
`NewAdmin(admin(address), newAdmin(address))`
`NewOperator(operator(address), newOperator(address))`
`OrderCancelled(orderHash(bytes32))`
`OrderFilled(orderHash(bytes32), maker(address), taker(address), makerAssetId(uint256), takerAssetId(uint256), making(uint256), taking(uint256), fee(uint256))`
`OrdersMatched(orderHash(bytes32), maker(address), makerAssetId(uint256), takerAssetId(uint256), making(uint256), taking(uint256))`
`ProxyFactoryUpdated(oldProxyFactory(address), newProxyFactory(address))`
`RemovedAdmin(admin(address), oldAdmin(address))`
`RemovedOperator(operator(address), oldOperator(address))`
`SafeFactoryUpdated(oldSafeFactory(address), newSafeFactory(address))`
`TokenRegistered(token0(uint256), token1(uint256), conditionId(bytes32))`
`TradingPaused(trader(address))`
`TradingUnpaused(trader(address))`
Conditional Tokens
`ConditionPreparation(conditionId(bytes32), oracle(address), questionId(bytes32), outcomeSlotCount(uint256))`
`ConditionResolution(conditionId(bytes32), oracle(address), questionId(bytes32), outcomeSlotCount(uint256), payoutNumerators(uint256[]))`
`PositionSplit(stakeholder(address), collateralToken(address), parentCollectionId(bytes32), conditionId(bytes32), partition(uint256[]), amount(uint256))`
`PositionsMerge(stakeholder(address), collateralToken(address), parentCollectionId(bytes32), conditionId(bytes32), partition(uint256[]), amount(uint256))`
`PayoutRedemption(redeemer(address), collateralToken(address), parentCollectionId(bytes32), conditionId(bytes32), indexSets(uint256[]), payout(uint256))`
`URI(value(string), id(uint256))`
Neg Risk Operator
`MarketPrepared(marketId(bytes32), feeBips(uint256), data(bytes))`

Table B.2 – continued from previous page

Contract / Event Signatures
NewAdmin(admin(address), newAdmin(address))
QuestionEmergencyResolved(questionId(bytes32), result(bool))
QuestionFlagged(questionId(bytes32))
QuestionPrepared(marketId(bytes32), questionId(bytes32), requestId(bytes32), questionIndex(uint256), data(bytes))
QuestionReported(questionId(bytes32), requestId(bytes32), result(bool))
QuestionResolved(questionId(bytes32), result(bool))
QuestionUnflagged(questionId(bytes32))
RemovedAdmin(admin(address), oldAdmin(address))
Neg Risk Adapter
MarketPrepared(marketId(bytes32), oracle(address), feeBips(uint256), data(bytes))
NewAdmin(admin(address), newAdmin(address))
OutcomeReported(marketId(bytes32), questionId(bytes32), outcome(bool))
PayoutRedemption(redeemer(address), conditionId(bytes32), amounts(uint256[]), payout(uint256))
PositionSplit(stakeholder(address), conditionId(bytes32), amount(uint256))
PositionsConverted(stakeholder(address), marketId(bytes32), indexSet(uint256), amount(uint256))
PositionsMerge(stakeholder(address), conditionId(bytes32), amount(uint256))
QuestionPrepared(marketId(bytes32), questionId(bytes32), index(uint256), data(bytes))
RemovedAdmin(admin(address), oldAdmin(address))
UmaCtfadapter
AncillaryDataUpdated(questionId(bytes32), creator(address), ancillaryData(bytes))
NewAdmin(admin(address), newAdmin(address))
QuestionFlagged(questionId(bytes32))
QuestionInitialized(questionId(bytes32), timestamp(uint256), creator(address), ancillaryData(bytes), rewardToken(address), reward(uint256), proposalBond(uint256))
QuestionManuallyResolved(questionId(bytes32), payouts(uint256[]))
QuestionPaused(questionId(bytes32))
QuestionReset(questionId(bytes32))

Table B.2 – continued from previous page

Contract / Event Signatures
QuestionResolved(questionId(bytes32), price(int256), payouts(uint256[]))
QuestionUnflagged(questionId(bytes32))
QuestionUnpaused(questionId(bytes32))
RemovedAdmin(admin(address), oldAdmin(address))
Managed Optimistic Oracle V2
MessageSent(data(bytes))
PriceRequestAdded(identifier(bytes32), timestamp(uint256), ancillaryData(bytes), childRequestId(bytes32))
PriceRequestBridged(requester(address), identifier(bytes32), time(uint256), ancillaryData(bytes), childRequestId(bytes32), parentRequestId(bytes32))
PushedPrice(identifier(bytes32), timestamp(uint256), ancillaryData(bytes), price(int256), childRequestId(bytes32))
ResolvedLegacyRequest(identifier(bytes32), timestamp(uint256), ancillaryData(bytes), price(int256), childRequestId(bytes32), parentRequestId(bytes32))
AllowedBondRangeUpdated(proposer(address), minBond(uint256), maxBond(uint256))
CustomBondSet(identifier(bytes32), proposer(address), ancillaryData(bytes32), data(bytes), currency(address), bond(uint256))
CustomLivenessSet(identifier(bytes32), proposer(address), ancillaryData(bytes32), data(bytes), liveness(uint256))
CustomProposerWhitelistSet(identifier(bytes32), proposer(address), ancillaryData(bytes32), data(bytes), newProposer(address))
DefaultAdminDelayChangeCanceled()
DefaultAdminDelayChangeScheduled(delay(uint48), newDelay(uint48))
DefaultAdminTransferCanceled()
DefaultAdminTransferScheduled(newAdmin(address), time(uint48))
DefaultProposerWhitelistUpdated(newWhitelist(address))
DisputePrice(requester(address), proposer(address), disputer(address), identifier(bytes32), timestamp(uint256), ancillaryData(bytes), proposedPrice(int256))
Initialized(version(uint64))
MinimumLivenessUpdated(minimumLiveness(uint256))

Table B.2 – continued from previous page

Contract / Event Signatures
ProposePrice(requester(address), proposer(address), identifier(bytes32), timestamp(uint256), ancillaryData(bytes), proposedPrice(int256), expirationTime(uint256), currency(address))
RequestManagerAdded(requestManager(address))
RequestManagerRemoved(requestManager(address))
RequestPrice(requester(address), identifier(bytes32), timestamp(uint256), ancillaryData(bytes), currency(address), reward(uint256), finalFee(uint256))
RequesterWhitelistUpdated(newWhitelist(address))
RoleAdminChanged(role(bytes32), previousAdminRole(bytes32), newAdminRole(bytes32))
RoleGranted(role(bytes32), account(address), adminRole(bytes32))
RoleRevoked(role(bytes32), account(address), adminRole(bytes32))
Settle(requester(address), proposer(address), disputer(address), identifier(bytes32), timestamp(uint256), ancillaryData(bytes), resolvedPrice(int256), payout(uint256))
Upgraded(implementation(address))
Oracle Child Tunnel
PriceRequestBridged(requester(address), identifier(bytes32), time(uint256), ancillaryData(bytes), childRequestId(bytes32), parentRequestId(bytes32))
ResolvedLegacyRequest(identifier(bytes32), timestamp(uint256), ancillaryData(bytes), price(int256), childRequestId(bytes32), parentRequestId(bytes32))
PriceRequestAdded(identifier(bytes32), timestamp(uint256), ancillaryData(bytes), childRequestId(bytes32))
PushedPrice(identifier(bytes32), timestamp(uint256), ancillaryData(bytes), price(int256), childRequestId(bytes32))
MessageSent(data(bytes))
FxTunnel
NewFxMessage(rootMessageSender(address), receiver(address), data(bytes))
Optimistic Oracle V2
RequestPrice(requester(address), identifier(bytes32), timestamp(uint256), ancillaryData(bytes), currency(address), reward(uint256), finalFee(uint256))

Table B.2 – continued from previous page

Contract / Event Signatures
`ProposePrice(requester(address), proposer(address), identifier(bytes32), timestamp(uint256), ancillaryData(bytes), proposedPrice(int256), expirationTime(uint256), currency(address))`
`DisputePrice(requester(address), proposer(address), disputer(address), identifier(bytes32), timestamp(uint256), ancillaryData(bytes), proposedPrice(int256))`
`Settle(requester(address), proposer(address), disputer(address), identifier(bytes32), timestamp(uint256), ancillaryData(bytes), resolvedPrice(int256), payout(uint256))`

Appendix C

Polymarket Event Object JSON Tree Analysis

Table C.1: JSON Tree Structure Analysis of `/events` Endpoint Responses. For each entry, the Path and Total Count are on the first line, with Presence and Type Makeup on the indented second line.

Path	Total Count
`ROOT` *Presence: 100.0%* *Type: Object (100.0)*	126558
`ROOT.tags` *Presence: 100.0%* *Type: Object (100.0)*	126555
`ROOT.tags.[]` *Presence: 100.0%* *Type: Array (100.0)*	126555
`ROOT.markets` *Presence: 100.0%* *Type: Object (100.0)*	126523
`ROOT.markets.[]` *Presence: 100.0%* *Type: Array (100.0)*	126523
`ROOT.series` *Presence: 83.6%* *Type: Object (100.0)*	105820
`ROOT.series.[]` *Presence: 83.6%* *Type: Array (100.0)*	105820
`ROOT.markets.[].clobRewards` *Presence: 7.7%* *Type: Object (100.0)*	9805
`ROOT.markets.[].clobRewards.[]` *Presence: 7.7%* *Type: Array (100.0)*	9805

Table C.1 – continued from previous page

Path	Total Count
`ROOT.eventCreators` *Presence: 0.1% Type: Object (100.0)*	87
`ROOT.eventCreators.[]` *Presence: 0.1% Type: Array (100.0)*	87
`ROOT.tags.[].slug` *Presence: 100.0% Type: str:general (100.0), str:integer (0.0)*	740323
`ROOT.tags.[].requiresTranslation` *Presence: 100.0% Type: bool (100.0)*	740323
`ROOT.tags.[].label` *Presence: 100.0% Type: str:general (100.0), str:integer (0.0)*	740323
`ROOT.tags.[].id` *Presence: 100.0% Type: str:integer (100.0)*	740323
`ROOT.tags.[].updatedAt` *Presence: 100.0% Type: str:timestamp (100.0)*	740162
`ROOT.tags.[].createdAt` *Presence: 98.8% Type: str:timestamp (100.0)*	731578
`ROOT.tags.[].forceShow` *Presence: 88.6% Type: bool (100.0)*	655610
`ROOT.tags.[].publishedAt` *Presence: 40.7% Type: str:general (100.0)*	301036
`ROOT.markets.[].approved` *Presence: 100.0% Type: bool (100.0)*	294240
`ROOT.markets.[].active` *Presence: 100.0% Type: bool (100.0)*	294240
`ROOT.markets.[].clearBookOnStart` *Presence: 100.0% Type: bool (100.0)*	294240
`ROOT.markets.[].createdAt` *Presence: 100.0% Type: str:timestamp (100.0)*	294240
`ROOT.markets.[].cyom` *Presence: 100.0% Type: bool (100.0)*	294240
`ROOT.markets.[].feesEnabled` *Presence: 100.0% Type: bool (100.0)*	294240
`ROOT.markets.[].umaResolutionStatuses` *Presence: 100.0% Type: str:json_ encoded (100.0)*	294240

Table C.1 – continued from previous page

Path	Total Count
`ROOT.markets.[].deploying` *Presence: 100.0% Type: bool (100.0)*	294240
`ROOT.markets.[].description` *Presence: 100.0% Type: str:general (100.0)*	294240
`ROOT.markets.[].funded` *Presence: 100.0% Type: bool (100.0)*	294240
`ROOT.markets.[].archived` *Presence: 100.0% Type: bool (100.0)*	294240
`ROOT.markets.[].bestAsk` *Presence: 100.0% Type: float (59.7), int (40.3)*	294240
`ROOT.markets.[].manualActivation` *Presence: 100.0% Type: bool (100.0)*	294240
`ROOT.markets.[].id` *Presence: 100.0% Type: str:integer (100.0)*	294240
`ROOT.markets.[].holdingRewardsEnabled` *Presence: 100.0% Type: bool (100.0)*	294240
`ROOT.markets.[].pagerDutyNotificationEnabled` *Presence: 100.0% Type: bool (100.0)*	294240
`ROOT.markets.[].pendingDeployment` *Presence: 100.0% Type: bool (100.0)*	294240
`ROOT.markets.[].question` *Presence: 100.0% Type: str:general (100.0)*	294240
`ROOT.markets.[].ready` *Presence: 100.0% Type: bool (100.0)*	294240
`ROOT.markets.[].rewardsMaxSpread` *Presence: 100.0% Type: int (82.9), float (17.1)*	294240
`ROOT.markets.[].outcomes` *Presence: 100.0% Type: str:json_ encoded (100.0)*	294240
`ROOT.markets.[].rfqEnabled` *Presence: 100.0% Type: bool (100.0)*	294240
`ROOT.markets.[].rewardsMinSize` *Presence: 100.0% Type: int (100.0), float (0.0)*	294240
`ROOT.markets.[].spread` *Presence: 100.0% Type: float (83.4), int (16.6)*	294240

Table C.1 – continued from previous page

Path	Total Count
`ROOT.markets.[].requiresTranslation` *Presence: 100.0%* *Type: bool (100.0)*	294240
`ROOT.markets.[].restricted` *Presence: 100.0%* *Type: bool (100.0)*	294240
`ROOT.markets.[].slug` *Presence: 100.0%* *Type: str:general (100.0)*	294240
`ROOT.markets.[].conditionId` *Presence: 100.0%* *Type: str:hexadecimal (99.97), str:empty (0.03)*	294240
`ROOT.markets.[].closed` *Presence: 100.0%* *Type: bool (100.0)*	294240
`ROOT.markets.[].negRiskOther` *Presence: 100.0%* *Type: bool (100.0)*	294240
`ROOT.markets.[].marketMakerAddress` *Presence: 100.0%* *Type: str:empty (91.1), str:hexadecimal (8.9)*	294240
`ROOT.markets.[].new` *Presence: 100.0%* *Type: bool (100.0)*	294209
`ROOT.markets.[].clobTokenIds` *Presence: 99.9%* *Type: str:json_ encoded (100.0)*	294080
`ROOT.markets.[].image` *Presence: 99.8%* *Type: str:general (99.6), str:empty (0.4)*	293596
`ROOT.markets.[].icon` *Presence: 99.7%* *Type: str:general (99.6), str:empty (0.4)*	293371
`ROOT.markets.[].updatedAt` *Presence: 99.6%* *Type: str:timestamp (100.0)*	293018
`ROOT.markets.[].hasReviewedDates` *Presence: 99.3%* *Type: bool (100.0)*	292264
`ROOT.markets.[].questionID` *Presence: 99.3%* *Type: str:hexadecimal (100.0)*	292192
`ROOT.markets.[].enableOrderBook` *Presence: 99.1%* *Type: bool (100.0)*	291620
`ROOT.markets.[].endDate` *Presence: 99.0%* *Type: str:timestamp (100.0)*	291388
`ROOT.markets.[].startDate` *Presence: 98.7%* *Type: str:timestamp (100.0)*	290272

Table C.1 – continued from previous page

Path	Total Count
`ROOT.markets.[].orderMinSize` *Presence: 98.4%* *Type: int (100.0), float (0.0)*	289522
`ROOT.markets.[].orderPriceMinTickSize` *Presence: 98.4%* *Type: float (100.0), int (0.0)*	289520
`ROOT.markets.[].endDateIso` *Presence: 98.2%* *Type: str:timestamp (100.0)*	288839
`ROOT.markets.[].acceptingOrders` *Presence: 97.0%* *Type: bool (100.0)*	285530
`ROOT.markets.[].negRisk` *Presence: 96.8%* *Type: bool (100.0)*	284707
`ROOT.markets.[].outcomePrices` *Presence: 95.5%* *Type: str:json_ encoded (100.0)*	280947
`ROOT.markets.[].automaticallyActive` *Presence: 95.4%* *Type: bool (100.0)*	280826
`ROOT.markets.[].featured` *Presence: 94.7%* *Type: bool (100.0)*	278588
`ROOT.markets.[].startDateIso` *Presence: 91.5%* *Type: str:timestamp (100.0)*	269084
`ROOT.markets.[].volume` *Presence: 91.4%* *Type: str:float (77.1), str:integer (22.9)*	268901
`ROOT.markets.[].volumeNum` *Presence: 91.4%* *Type: float (77.0), int (23.0)*	268901
`ROOT.markets.[].acceptingOrdersTimestamp` *Presence: 90.4%* *Type: str:timestamp (100.0)*	266051
`ROOT.markets.[].volumeClob` *Presence: 89.8%* *Type: float (76.6), int (23.4)*	264197
`ROOT.markets.[].groupItemThreshold` *Presence: 89.6%* *Type: str:integer (100.0)*	263602
`ROOT.markets.[].closedTime` *Presence: 88.3%* *Type: str:general (100.0)*	259848
`ROOT.markets.[].lastTradePrice` *Presence: 88.0%* *Type: int (78.0), float (22.0)*	258812
`ROOT.markets.[].umaResolutionStatus` *Presence: 87.6%* *Type: str:general (100.0)*	257686

Table C.1 – continued from previous page

Path	Total Count
`ROOT.markets.[].umaEndDate` *Presence: 87.4%* *Type: str:timestamp (96.9), str:general (3.1)*	257309
`ROOT.markets.[].automaticallyResolved` *Presence: 84.7%* *Type: bool (100.0)*	249301
`ROOT.tags.[].updatedBy` *Presence: 32.7%* *Type: int (100.0)*	241730
`ROOT.markets.[].resolvedBy` *Presence: 81.8%* *Type: str:hexadecimal (100.0)*	240654
`ROOT.markets.[].resolutionSource` *Presence: 80.4%* *Type: str:general (74.5), str:empty (25.5)*	236453
`ROOT.tags.[].isCarousel` *Presence: 31.2%* *Type: bool (100.0)*	231335
`ROOT.markets.[].volume1yrClob` *Presence: 77.1%* *Type: float (61.6), int (38.4)*	226919
`ROOT.markets.[].volume1yr` *Presence: 77.1%* *Type: float (61.6), int (38.4)*	226919
`ROOT.markets.[].volume1moClob` *Presence: 77.0%* *Type: float (61.5), int (38.5)*	226441
`ROOT.markets.[].volume1mo` *Presence: 77.0%* *Type: float (61.5), int (38.5)*	226441
`ROOT.markets.[].submitted_by` *Presence: 76.7%* *Type: str:hexadecimal (99.7), others (0.3)*	225800
`ROOT.markets.[].volume1wkClob` *Presence: 76.0%* *Type: float (60.6), int (39.4)*	223602
`ROOT.markets.[].volume1wk` *Presence: 76.0%* *Type: float (60.6), int (39.4)*	223602
`ROOT.markets.[].umaBond` *Presence: 74.5%* *Type: str:integer (98.7), str:float (1.3)*	219274
`ROOT.markets.[].umaReward` *Presence: 74.5%* *Type: str:integer (98.6), str:float (1.4)*	219274
`ROOT.markets.[].negRiskRequestID` *Presence: 70.9%* *Type: str:empty (54.9), str:hexadecimal (45.1)*	208566
`ROOT.markets.[].groupItemTitle` *Presence: 69.9%* *Type: str:general (91.6), others (8.4)*	205672

Table C.1 – continued from previous page

Path	Total Count
`ROOT.markets.[].oneDayPriceChange` *Presence: 68.4%* *Type: float (75.7), int (24.3)*	201115
`ROOT.markets.[].deployingTimestamp` *Presence: 64.6%* *Type: str:timestamp (100.0)*	189939
`ROOT.markets.[].customLiveness` *Presence: 61.3%* *Type: int (100.0)*	180461
`ROOT.markets.[].bestBid` *Presence: 58.7%* *Type: float (71.6), int (28.4)*	172631
`ROOT.markets.[].oneHourPriceChange` *Presence: 53.9%* *Type: float (52.6), int (47.4)*	158604
`ROOT.markets.[].competitive` *Presence: 47.1%* *Type: int (85.3), float (14.7)*	138520
`ROOT.markets.[].liquidity` *Presence: 47.1%* *Type: str:integer (80.6), str:float (19.4)*	138485
`ROOT.markets.[].liquidityNum` *Presence: 47.1%* *Type: int (82.3), float (17.7)*	138485
`ROOT.markets.[].showGmpOutcome` *Presence: 46.8%* *Type: bool (100.0)*	137594
`ROOT.markets.[].showGmpSeries` *Presence: 46.8%* *Type: bool (100.0)*	137593
`ROOT.markets.[].liquidityClob` *Presence: 45.4%* *Type: int (84.3), float (15.7)*	133631
`ROOT.markets.[].oneWeekPriceChange` *Presence: 42.4%* *Type: int (60.2), float (39.8)*	124819
`ROOT.active` *Presence: 100.0%* *Type: bool (100.0)*	126558
`ROOT.title` *Presence: 100.0%* *Type: str:general (100.0)*	126558
`ROOT.pendingDeployment` *Presence: 100.0%* *Type: bool (100.0)*	126558
`ROOT.requiresTranslation` *Presence: 100.0%* *Type: bool (100.0)*	126558
`ROOT.negRiskAugmented` *Presence: 100.0%* *Type: bool (100.0)*	126558

Table C.1 – continued from previous page

Path	Total Count
`ROOT.image`	126558
Presence: 100.0% *Type: str:general (99.8), str:empty (0.2)*	
`ROOT.icon`	126558
Presence: 100.0% *Type: str:general (99.8), str:empty (0.2)*	
`ROOT.id`	126558
Presence: 100.0% *Type: str:integer (100.0)*	
`ROOT.deploying`	126558
Presence: 100.0% *Type: bool (100.0)*	
`ROOT.closed`	126558
Presence: 100.0% *Type: bool (100.0)*	
`ROOT.cyom`	126558
Presence: 100.0% *Type: bool (100.0)*	
`ROOT.createdAt`	126558
Presence: 100.0% *Type: str:timestamp (100.0)*	
`ROOT.enableNegRisk`	126558
Presence: 100.0% *Type: bool (100.0)*	
`ROOT.showAllOutcomes`	126558
Presence: 100.0% *Type: bool (100.0)*	
`ROOT.showMarketImages`	126558
Presence: 100.0% *Type: bool (100.0)*	
`ROOT.slug`	126558
Presence: 100.0% *Type: str:general (100.0)*	
`ROOT.ticker`	126556
Presence: 100.0% *Type: str:general (100.0)*	
`ROOT.archived`	126550
Presence: 100.0% *Type: bool (100.0)*	
`ROOT.restricted`	126550
Presence: 100.0% *Type: bool (100.0)*	
`ROOT.featured`	126534
Presence: 100.0% *Type: bool (100.0)*	
`ROOT.startDate`	126529
Presence: 100.0% *Type: str:timestamp (100.0)*	
`ROOT.new`	126520
Presence: 100.0% *Type: bool (100.0)*	

Table C.1 – continued from previous page

Path	Total Count
`ROOT.updatedAt` *Presence: 100.0%* *Type: str:timestamp (100.0)*	126495
`ROOT.creationDate` *Presence: 99.9%* *Type: str:timestamp (100.0)*	126461
`ROOT.endDate` *Presence: 99.9%* *Type: str:timestamp (100.0)*	126388
`ROOT.description` *Presence: 99.5%* *Type: str:general (100.0), others (0.0)*	125876
`ROOT.commentCount` *Presence: 99.3%* *Type: int (100.0)*	125633
`ROOT.openInterest` *Presence: 98.7%* *Type: int (100.0)*	124935
`ROOT.enableOrderBook` *Presence: 97.5%* *Type: bool (100.0)*	123449
`ROOT.closedTime` *Presence: 94.8%* *Type: str:timestamp (100.0)*	119928
`ROOT.automaticallyResolved` *Presence: 90.5%* *Type: bool (100.0)*	114539
`ROOT.negRisk` *Presence: 89.2%* *Type: bool (100.0)*	112947
`ROOT.resolutionSource` *Presence: 88.3%* *Type: str:general (87.2), str:empty (12.8)*	111781
`ROOT.seriesSlug` *Presence: 83.7%* *Type: str:general (100.0)*	105868
`ROOT.series.[].id` *Presence: 100.0%* *Type: str:integer (100.0)*	105820
`ROOT.series.[].closed` *Presence: 100.0%* *Type: bool (100.0)*	105820
`ROOT.series.[].createdAt` *Presence: 100.0%* *Type: str:timestamp (100.0)*	105820
`ROOT.series.[].requiresTranslation` *Presence: 100.0%* *Type: bool (100.0)*	105820
`ROOT.series.[].commentCount` *Presence: 100.0%* *Type: int (100.0)*	105820

Table C.1 – continued from previous page

Path	Total Count
`ROOT.series.[].active` *Presence: 100.0% Type: bool (100.0)*	105820
`ROOT.series.[].archived` *Presence: 100.0% Type: bool (100.0)*	105820
`ROOT.series.[].ticker` *Presence: 100.0% Type: str:general (100.0)*	105820
`ROOT.series.[].slug` *Presence: 100.0% Type: str:general (100.0)*	105820
`ROOT.series.[].title` *Presence: 100.0% Type: str:general (100.0)*	105820
`ROOT.series.[].updatedAt` *Presence: 100.0% Type: str:timestamp (100.0)*	105820
`ROOT.series.[].seriesType` *Presence: 99.9% Type: str:general (100.0)*	105750
`ROOT.series.[].recurrence` *Presence: 99.9% Type: str:general (99.9), str:empty (0.1)*	105750
`ROOT.volume` *Presence: 82.8% Type: float (95.9), int (4.1)*	104840
`ROOT.automaticallyActive` *Presence: 81.5% Type: bool (100.0)*	103195
`ROOT.markets.[].secondsDelay` *Presence: 34.2% Type: int (100.0)*	100673
`ROOT.markets.[].gameStartTime` *Presence: 33.4% Type: str:general (100.0)*	98324
`ROOT.markets.[].negRiskMarketID` *Presence: 32.0% Type: str:hexadecimal (100.0)*	94024
`ROOT.markets.[].oneMonthPriceChange` *Presence: 30.5% Type: int (83.6), float (16.4)*	89813
`ROOT.startTime` *Presence: 69.7% Type: str:timestamp (100.0)*	88272
`ROOT.series.[].featured` *Presence: 79.2% Type: bool (100.0)*	83843
`ROOT.series.[].restricted` *Presence: 78.7% Type: bool (100.0)*	83242

Table C.1 – continued from previous page

Path	Total Count
`ROOT.series.[].image` *Presence: 76.6%* *Type: str:empty (58.4), str:general (41.6)*	81104
`ROOT.series.[].icon` *Presence: 76.6%* *Type: str:empty (58.4), str:general (41.6)*	81104
`ROOT.markets.[].eventStartTime` *Presence: 26.5%* *Type: str:timestamp (100.0)*	78103
`ROOT.tags.[].createdBy` *Presence: 10.4%* *Type: int (100.0)*	76858
`ROOT.series.[].liquidity` *Presence: 71.8%* *Type: float (100.0)*	75989
`ROOT.series.[].volume` *Presence: 71.6%* *Type: float (98.6), int (1.4)*	75775
`ROOT.markets.[].oneYearPriceChange` *Presence: 25.6%* *Type: int (99.9), float (0.1)*	75186
`ROOT.markets.[].volume1wkAmm` *Presence: 25.5%* *Type: int (100.0)*	75118
`ROOT.markets.[].volume1yrAmm` *Presence: 25.5%* *Type: int (100.0)*	75118
`ROOT.markets.[].volume1moAmm` *Presence: 25.5%* *Type: int (100.0)*	75118
`ROOT.volume1yr` *Presence: 58.7%* *Type: float (77.3), int (22.7)*	74284
`ROOT.volume1mo` *Presence: 58.7%* *Type: float (77.3), int (22.7)*	74274
`ROOT.volume1wk` *Presence: 58.5%* *Type: float (77.1), int (22.9)*	74043
`ROOT.markets.[].sportsMarketType` *Presence: 22.9%* *Type: str:general (100.0)*	67474
`ROOT.tags.[].forceHide` *Presence: 8.2%* *Type: bool (100.0)*	60671
`ROOT.competitive` *Presence: 47.9%* *Type: int (92.0), float (8.0)*	60616
`ROOT.liquidity` *Presence: 47.8%* *Type: int (91.0), float (9.0)*	60435

Table C.1 – continued from previous page

Path	Total Count
`ROOT.liquidityClob` *Presence: 47.7% Type: int (91.1), float (8.9)*	60431
`ROOT.markets.[].volume24hr` *Presence: 18.9% Type: int (85.6), float (14.4)*	55570
`ROOT.liquidityAmm` *Presence: 43.5% Type: int (99.9), float (0.1)*	55028
`ROOT.markets.[].volume24hrClob` *Presence: 17.3% Type: int (84.3), float (15.7)*	50866
`ROOT.markets.[].line` *Presence: 14.8% Type: float (98.3), int (1.7)*	43542
`ROOT.markets.[].seriesColor` *Presence: 14.1% Type: str:empty (99.0), str:general (1.0)*	41612
`ROOT.markets.[].volumeAmm` *Presence: 14.1% Type: int (98.7), float (1.3)*	41454
`ROOT.markets.[].volume24hrAmm` *Presence: 14.0% Type: int (100.0)*	41306
`ROOT.markets.[].clobRewards.[].rewardsAmount` *Presence: 100.0% Type: int (100.0)*	32054
`ROOT.markets.[].clobRewards.[].id` *Presence: 100.0% Type: str:integer (100.0)*	32054
`ROOT.markets.[].clobRewards.[].endDate` *Presence: 100.0% Type: str:timestamp (100.0)*	32054
`ROOT.markets.[].clobRewards.[].conditionId` *Presence: 100.0% Type: str:hexadecimal (100.0)*	32054
`ROOT.markets.[].clobRewards.[].assetAddress` *Presence: 100.0% Type: str:hexadecimal (100.0)*	32054
`ROOT.markets.[].clobRewards.[].startDate` *Presence: 100.0% Type: str:timestamp (100.0)*	32054
`ROOT.markets.[].clobRewards.[].rewardsDailyRate` *Presence: 100.0% Type: int (100.0), float (0.0)*	32054
`ROOT.eventDate` *Presence: 21.4% Type: str:timestamp (100.0)*	27116
`ROOT.markets.[].fpmmLive` *Presence: 9.0% Type: bool (100.0)*	26535

Table C.1 – continued from previous page

Path	Total Count
`ROOT.markets.[].wideFormat` *Presence: 8.8%* *Type: bool (100.0)*	25839
`ROOT.markets.[].fee` *Presence: 8.8%* *Type: str:integer (100.0), str:float (0.0)*	25829
`ROOT.eventWeek` *Presence: 18.6%* *Type: int (100.0)*	23516
`ROOT.markets.[].readyForCron` *Presence: 7.8%* *Type: bool (100.0)*	22819
`ROOT.period` *Presence: 16.6%* *Type: str:general (94.1), str:empty (5.9)*	20964
`ROOT.markets.[].sentDiscord` *Presence: 7.0%* *Type: bool (100.0)*	20529
`ROOT.ended` *Presence: 16.1%* *Type: bool (100.0)*	20316
`ROOT.live` *Presence: 16.0%* *Type: bool (100.0)*	20290
`ROOT.score` *Presence: 15.1%* *Type: str:general (99.6), str:empty (0.4)*	19084
`ROOT.markets.[].notificationsEnabled` *Presence: 6.2%* *Type: bool (100.0)*	18372
`ROOT.sortBy` *Presence: 14.4%* *Type: str:general (100.0)*	18208
`ROOT.gameId` *Presence: 13.5%* *Type: int (100.0)*	17121
`ROOT.elapsed` *Presence: 13.4%* *Type: str:empty (96.5), others (3.5)*	16898
`ROOT.finishedTimestamp` *Presence: 10.9%* *Type: str:timestamp (100.0)*	13752
`ROOT.negRiskMarketID` *Presence: 9.2%* *Type: str:hexadecimal (99.9), str:empty (0.1)*	11614
`ROOT.series.[].new` *Presence: 10.1%* *Type: bool (100.0)*	10705
`ROOT.markets.[].gameId` *Presence: 3.5%* *Type: str:hexadecimal (55.1), str:integer (44.9)*	10354

Table C.1 – continued from previous page

Path	Total Count
`ROOT.markets.[].marketType` *Presence: 3.4% Type: str:general (100.0)*	10072
`ROOT.markets.[].creator` *Presence: 3.4% Type: str:empty (100.0)*	10072
`ROOT.deployingTimestamp` *Presence: 6.7% Type: str:timestamp (100.0)*	8511
`ROOT.volume24hr` *Presence: 6.2% Type: int (65.4), float (34.6)*	7886
`ROOT.series.[].layout` *Presence: 7.4% Type: str:general (100.0)*	7868
`ROOT.series.[].volume24hr` *Presence: 7.4% Type: int (100.0)*	7816
`ROOT.series.[].publishedAt` *Presence: 7.4% Type: str:general (100.0)*	7816
`ROOT.series.[].competitive` *Presence: 7.4% Type: str:integer (100.0)*	7816
`ROOT.series.[].commentsEnabled` *Presence: 7.4% Type: bool (100.0)*	7816
`ROOT.series.[].updatedBy` *Presence: 7.3% Type: str:integer (100.0)*	7710
`ROOT.series.[].createdBy` *Presence: 7.3% Type: str:integer (100.0)*	7710
`ROOT.series.[].startDate` *Presence: 6.6% Type: str:timestamp (100.0)*	7013
`ROOT.gmpChartMode` *Presence: 5.4% Type: str:general (100.0)*	6866
`ROOT.homeTeamName` *Presence: 4.6% Type: str:general (100.0)*	5770
`ROOT.awayTeamName` *Presence: 4.6% Type: str:general (100.0)*	5770
`ROOT.markets.[].commentsEnabled` *Presence: 2.0% Type: bool (100.0)*	5772
`ROOT.markets.[].updatedBy` *Presence: 1.9% Type: int (100.0)*	5544

Table C.1 – continued from previous page

Path	Total Count
`ROOT.markets.[].umaEndDateIso` *Presence: 1.7%* *Type: str:timestamp (100.0)*	5124
`ROOT.published_at` *Presence: 4.0%* *Type: str:general (100.0)*	5089
`ROOT.markets.[].category` *Presence: 1.5%* *Type: str:general (99.9), str:empty (0.1)*	4353
`ROOT.markets.[].takerBaseFee` *Presence: 1.1%* *Type: int (100.0)*	3313
`ROOT.markets.[].makerBaseFee` *Presence: 1.1%* *Type: int (100.0)*	3312
`ROOT.markets.[].twitterCardLocation` *Presence: 1.0%* *Type: str:general (100.0)*	3026
`ROOT.markets.[].twitterCardLastRefreshed` *Presence: 1.0%* *Type: str:integer (100.0)*	3003
`ROOT.category` *Presence: 2.2%* *Type: str:general (100.0)*	2837
`ROOT.markets.[].mailchimpTag` *Presence: 0.9%* *Type: str:integer (90.0), str:null_like (10.0)*	2668
`ROOT.commentsEnabled` *Presence: 1.8%* *Type: bool (100.0)*	2246
`ROOT.markets.[].marketGroup` *Presence: 0.7%* *Type: int (100.0)*	2092
`ROOT.parentEventId` *Presence: 1.3%* *Type: int (100.0)*	1603
`ROOT.markets.[].twitterCardLastValidated` *Presence: 0.5%* *Type: str:float (100.0)*	1435
`ROOT.negRiskFeeBips` *Presence: 0.8%* *Type: int (100.0)*	999
`ROOT.series.[].description` *Presence: 0.6%* *Type: str:general (100.0)*	628
`ROOT.updatedBy` *Presence: 0.4%* *Type: str:integer (100.0)*	530
`ROOT.featuredImage` *Presence: 0.4%* *Type: str:general (66.2), str:empty (33.8)*	518

Table C.1 – continued from previous page

Path	Total Count
`ROOT.series.[].cgAssetName` *Presence: 0.5% Type: str:general (100.0)*	512
`ROOT.series.[].pythTokenID` *Presence: 0.5% Type: str:hexadecimal (100.0)*	512
`ROOT.series.[].subtitle` *Presence: 0.5% Type: str:general (100.0)*	512
`ROOT.featuredOrder` *Presence: 0.3% Type: int (100.0)*	373
`ROOT.markets.[].categoryMailchimpTag` *Presence: 0.1% Type: str:null_like (83.1), str:integer (16.9)*	308
`ROOT.createdBy` *Presence: 0.1% Type: str:integer (100.0)*	182
`ROOT.countryName` *Presence: 0.1% Type: str:general (99.3), str:empty (0.7)*	140
`ROOT.electionType` *Presence: 0.1% Type: str:general (100.0)*	137
`ROOT.tweetCount` *Presence: 0.1% Type: int (100.0)*	134
`ROOT.totalsMainLine` *Presence: 0.1% Type: float (100.0)*	99
`ROOT.spreadsMainLine` *Presence: 0.1% Type: float (100.0)*	99
`ROOT.eventCreators.[].id` *Presence: 100.0% Type: str:integer (100.0)*	87
`ROOT.eventCreators.[].creatorImage` *Presence: 100.0% Type: str:general (100.0)*	87
`ROOT.eventCreators.[].creatorHandle` *Presence: 100.0% Type: str:general (100.0)*	87
`ROOT.eventCreators.[].creatorUrl` *Presence: 100.0% Type: str:general (100.0)*	87
`ROOT.eventCreators.[].creatorName` *Presence: 100.0% Type: str:general (100.0)*	87
`ROOT.eventCreators.[].createdAt` *Presence: 100.0% Type: str:timestamp (100.0)*	87

Table C.1 – continued from previous page

Path	Total Count
`ROOT.markets.[].subcategory` *Presence: 0.0%* *Type: str:general (89.2), str:empty (10.8)*	74
`ROOT.markets.[].lowerBound` *Presence: 0.0%* *Type: str:integer (78.1), str:float (19.2), others (2.7)*	73
`ROOT.markets.[].upperBound` *Presence: 0.0%* *Type: str:integer (79.2), str:float (19.4), others (1.4)*	72
`ROOT.markets.[].disqusThread` *Presence: 0.0%* *Type: str:general (100.0)*	68
`ROOT.subcategory` *Presence: 0.1%* *Type: str:general (98.5), str:empty (1.5)*	68
`ROOT.markets.[].formatType` *Presence: 0.0%* *Type: str:general (100.0)*	66
`ROOT.markets.[].twitterCardImage` *Presence: 0.0%* *Type: str:general (100.0)*	62
`ROOT.sportsradarMatchId` *Presence: 0.0%* *Type: str:general (100.0)*	47
`ROOT.turnProviderId` *Presence: 0.0%* *Type: str:integer (100.0)*	35
`ROOT.disqusThread` *Presence: 0.0%* *Type: str:general (100.0)*	28
`ROOT.gameStatus` *Presence: 0.0%* *Type: str:general (100.0)*	23
`ROOT.markets.[].groupItemRange` *Presence: 0.0%* *Type: str:json_ encoded (100.0)*	22
`ROOT.color` *Presence: 0.0%* *Type: str:general (95.2), str:integer (4.8)*	21
`ROOT.estimateValue` *Presence: 0.0%* *Type: bool (100.0)*	4
`ROOT.cantEstimate` *Presence: 0.0%* *Type: bool (100.0)*	4
`ROOT.carouselMap` *Presence: 0.0%* *Type: str:json_ encoded (100.0)*	4
`ROOT.markets.[].teamBID` *Presence: 0.0%* *Type: str:integer (100.0)*	4

Table C.1 – continued from previous page

Path	**Total Count**
`ROOT.markets.[].teamAID` *Presence: 0.0%* *Type: str:integer (100.0)*	4
`ROOT.estimatedValue` *Presence: 0.0%* *Type: str:float (100.0)*	3
`ROOT.markets.[].sponsorImage` *Presence: 0.0%* *Type: str:empty (100.0)*	2
`ROOT.markets.[].createdBy` *Presence: 0.0%* *Type: int (100.0)*	1
`ROOT.markets.[].denominationToken` *Presence: 0.0%* *Type: str:empty (100.0)*	1

Table C.2: Raw Pairwise Comparison Statistics for CLOB Market Timestamps

T_1	T_2	Pairs (N)	$P(T_1 < T_2)$	$P(T_1 = T_2)$	$P(T_1 > T_2)$
`acceptingOrdersTimestamp`	`closedTime`	233 087	100.00	0.00	0.00
`acceptingOrdersTimestamp`	`createdAt`	266 051	0.00	0.00	100.00
`acceptingOrdersTimestamp`	`endDate`	263 251	98.36	0.00	1.64
`acceptingOrdersTimestamp`	`startDate`	263 888	99.69	0.00	0.31
`acceptingOrdersTimestamp`	`umaEndDate`	233 022	100.00	0.00	0.00
`acceptingOrdersTimestamp`	`updatedAt`	266 051	100.00	0.00	0.00
`closedTime`	`createdAt`	259 848	0.00	0.00	100.00
`closedTime`	`endDate`	258 742	16.98	0.00	83.02
`closedTime`	`startDate`	256 120	0.02	0.00	99.98
`closedTime`	`umaEndDate`	257 286	1.06	98.07	0.87
`closedTime`	`updatedAt`	259 848	99.99	0.01	0.00
`createdAt`	`endDate`	291 388	97.87	0.00	2.13
`createdAt`	`startDate`	290 272	93.83	0.19	5.98
`createdAt`	`umaEndDate`	257 297	99.90	0.00	0.10
`createdAt`	`updatedAt`	293 018	100.00	0.00	0.00
`endDate`	`startDate`	287 460	2.24	0.72	97.04
`endDate`	`umaEndDate`	256 223	82.24	0.88	16.89
`endDate`	`updatedAt`	290 166	77.45	0.00	22.55
`startDate`	`umaEndDate`	254 373	99.21	0.77	0.02
`startDate`	`updatedAt`	289 086	99.58	0.42	0.00
`umaEndDate`	`updatedAt`	257 297	99.91	0.00	0.09