The Polymarket API: Architecture, Endpoints, and Use Cases

43 min read · Jan 22, 2026

Jung-Hua Liu Follow

Abstract

Polymarket is a decentralized prediction markets platform that enables users to trade on the outcomes of future events using blockchain technology*[1]*. This paper provides a comprehensive technical overview of the latest Polymarket API (as of early 2026), detailing its architecture, endpoints, and data structures in a formal documentation style. We describe how the API is organized into multiple services (notably the **Gamma** market data API, the **CLOB** trading API, and the **Data** API) and how each interacts with Polymarket's underlying smart contracts on the Polygon blockchain. Every publicly available REST endpoint and its fields are explicated with formal definitions, and we illustrate common usage patterns — such as fetching market listings, querying outcome prices, interpreting market liquidity, and deriving predictions from market data. Code examples in Python and JavaScript are provided to demonstrate integrating the API for tasks like retrieving markets and order books, computing probabilities, and analyzing liquidity. We also discuss how the Polymarket API ties into the on-chain infrastructure (e.g., conditional tokens and the UMA Optimistic Oracle for resolution) to ensure data integrity. This work serves as an academic-style reference for blockchain developers and researchers interested in leveraging Polymarket's API for building prediction market applications or conducting data-driven analysis of decentralized event markets.

Introduction

Prediction markets allow participants to trade shares in the outcomes of future events, with market prices reflecting the crowd's probability estimates. **Polymarket**

is a prominent blockchain-based prediction market platform where users trade on event outcomes using cryptocurrency (specifically, collateral like USDC on Polygon) *[1][2]*. Polymarket's design leverages smart contracts to mint outcome tokens for each event, an off-chain matching engine for efficient trading, and a decentralized oracle for resolving outcomes. To enable programmatic access to its markets, Polymarket provides a **comprehensive API** that developers and analysts can use to retrieve live market data, event listings, price information, order book snapshots, user positions, and more*[3][4]*. The latest version of this API (often referred to as the *Gamma* API for market data, alongside other components) offers a robust interface for building applications and conducting research on prediction markets.

This paper presents a full technical overview of the Polymarket API's latest iteration. We begin with background on Polymarket's system architecture, including how on-chain conditional tokens and oracle mechanisms underpin the markets. We then describe the API architecture, explaining the separation of concerns between the **Gamma API** (for market metadata and discovery), the **CLOB API** (central limit order book for trading operations), and the **Data API** (for user-specific data like positions and trade history). In a formal reference-style section, we enumerate every major API endpoint and provide detailed descriptions of each field returned, effectively serving as a structured documentation of the API. Subsequently, we explore common use cases and provide annotated code examples demonstrating how to use the API for tasks such as fetching the list of active markets, querying the latest prices of outcomes, examining the liquidity and order book depth of a market, and interpreting those results to make predictions or inform trading strategies. We also explain how the API's outputs relate to on-chain data — such as how an outcome's price corresponds to token swaps on the blockchain and how market resolution is achieved via the UMA Optimistic Oracle*[5]*. We conclude with a summary of the API's significance for developers and potential future directions.

Throughout the paper, we maintain a formal tone and cite relevant documentation and sources for clarity. The aim is to provide blockchain developers and researchers with an authoritative guide to the Polymarket API, enabling them to integrate Polymarket data into their applications or analyses with confidence.

Background

Polymarket Platform and On-Chain Mechanics

Polymarket operates on the Polygon (Ethereum) blockchain, using smart contracts to create and manage prediction markets*[2]*. Each market is defined by a **question** (e.g., "Will X happen by date Y?") and has a set of **outcomes** (for binary markets, typically "Yes" and "No"). Under the hood, Polymarket leverages a Conditional Token Framework (CTF) to represent outcome shares: when a market is created, collateral (usually USDC) is used to mint outcome tokens (often ERC-1155 tokens) for each possible outcome*[6]*. For instance, in a binary market, a Yes-token and a No-token are created, each redeemable for 1 USDC if its outcome is the true result. These tokens are freely tradable until the event outcome is resolved.

A key innovation in Polymarket's current architecture is its **Central Limit Order Book (CLOB)** system for trading outcome tokens. Earlier prediction markets often relied on automated market makers, but Polymarket now employs a hybrid-decentralized CLOB: an off-chain operator hosts an order book and matches orders, while final settlement occurs on-chain via smart contracts*[7]*. Participants submit signed orders (using the EIP-712 standard for typed data signing) to buy or sell outcome tokens at specified prices. The Polymarket exchange smart contract (deployed on Polygon) accepts matched orders from the operator and performs atomic swaps of outcome tokens for collateral, thereby settling trades non-custodially on-chain*[6]*. This design achieves low-latency trading with blockchain-enforced settlement. The exchange contract has been audited for security*[8],* and users retain control: if necessary, a user can cancel their orders on-chain independently, ensuring trust-minimization*[9]*.

Another crucial component is the **resolution oracle**. Polymarket integrates with UMA Protocol's *Optimistic Oracle* to settle markets in a decentralized manner*[5]*. When a market's end date is reached, the outcome is determined by asking UMA's oracle; anyone can propose the result (along with references/evidence in the ancillary data), which is assumed correct if not disputed within a challenge window*[5]*. A custom adapter contract called UmaCtfAdapter connects Polymarket's conditional tokens to UMA's oracle, ensuring that once the oracle finalizes an outcome, the corresponding outcome tokens become redeemable for collateral*[5] [10]*. This mechanism allows Polymarket to resolve arbitrary real-world questions in a trustless way. The resolution process, including possible disputes and settlement, happens on-chain via UMA's contracts, and the Polymarket API surfaces the resolution status (e.g., a field indicating if a market is resolved or still pending) to users.

In summary, Polymarket's on-chain infrastructure consists of smart contracts for market creation (minting and redeeming outcome tokens), an exchange contract for trading, and oracle adapters for resolution*[11]*. These are deployed on Polygon for scalability and cost efficiency. The on-chain events (such as trades and resolutions) are indexed by Polymarket's backend and also made available through public subgraphs and data services*[12][13]*. The Polymarket API builds on this foundation by providing convenient endpoints to query market data and interact with the trading system without requiring developers to parse low-level blockchain events directly.

Evolution to the Gamma API (Latest Version)

Polymarket's API has evolved over time to its current "Gamma" version. In earlier iterations of the platform, market data might have been accessed through different or more limited interfaces (including direct subgraph queries or ad-hoc endpoints). The **Gamma API**, introduced as a dedicated market data service, indexes all relevant on-chain and off-chain data and presents it in a developer-friendly REST format*[14]*. The term "Gamma" in Polymarket's documentation refers to this hosted service that provides enriched market metadata on top of the raw on-chain data, such as human-readable event information, categorized listings, and aggregated statistics*[14]*. By naming it Gamma, Polymarket indicates a mature third iteration of its market data API (following presumably earlier alpha/beta phases), though the documentation primarily uses "Gamma" to distinguish the market metadata API from other components.

At the time of writing, the Polymarket API suite is divided into several distinct segments, each with a specific role (detailed in the next section). The latest version has been refined with features like comprehensive filtering and sorting for market queries, real-time WebSocket feeds, and extended field sets (e.g., 24h/7d volume, open interest, best bid/ask, etc.) to support advanced use cases*[15][16]*. Continuous updates are documented in Polymarket's changelog, reflecting improvements such as new endpoints (for batch order placement, liquidity rewards data, etc.) and new fields in responses (for example, inclusion of min_order_size, tick_size, and neg_risk flags in order book data)*[15]*. These enhancements underscore Polymarket's commitment to providing a rich dataset for market analytics and automated trading.

For blockchain developers, the significance of the Gamma API and its related services is that they abstract away direct blockchain queries. Instead of manually

gathering events (e.g., using The Graph or Bitquery to get trades or resolutions*[13]*), developers can call REST endpoints to get structured JSON data about markets, prices, and user accounts. This greatly simplifies integration, making Polymarket's on-chain data accessible via familiar web API patterns.

API Architecture

The Polymarket API is organized into multiple components, each with its own base URL and purpose*[17]*. This modular architecture mirrors the platform's internal separation of concerns: market discovery and metadata are handled by one service, while trading-specific operations and user data are handled by others. Below, we outline the main API segments and their roles:

- **Gamma API (Market Metadata & Discovery)** — The Gamma API provides read-only access to market information: events, markets, tags, categories, and related metadata*[17]*. Its base endpoint is https://gamma-api.polymarket.com. This API allows developers to discover what markets exist, retrieve their details (questions, outcomes, status, etc.), and see aggregated stats like volume or liquidity. It indexes on-chain data (such as the market's condition ID and resolution info) and enriches it with off-chain metadata (titles, categories, images, etc.)*[14]*. For example, one can list all events or markets, filter them by active status or tags, and retrieve nested data like the outcomes and current prices. The Gamma API is public and does not require authentication for reads.

- **CLOB API (Central Limit Order Book & Trading)** — The CLOB API deals with live price data and order management on the Polymarket exchange. It is accessed via https://clob.polymarket.com*[17]*. This service exposes endpoints to get current order book states, price quotes, and to submit or cancel orders. Key read endpoints include fetching the current best bid/ask price for a given outcome token (GET /price), retrieving the full order book for a market (GET /book), or the midpoint price (GET /midpoint) among others*[18]*. Write operations, such as placing a new order (POST /order) or canceling an order (DELETE /order), require API keys and user authentication because they modify trading state*[19]*. The CLOB API is what market maker bots and trading clients would use to interact with Polymarket's order book in real time. It bridges to the on-chain exchange: orders placed via this API are matched off-chain and then settled on-chain by the Polymarket operator as described earlier. The CLOB API ensures that developers can programmatically trade, just as one would on a centralized exchange API, but with the security of on-chain settlement.

- **Data API (User Positions & History)** — Polymarket's Data API (https://data-api.polymarket.com) provides endpoints focused on user-specific data: positions, trade history, and activity logs*[20][21]*. This includes endpoints like GET /positions to fetch a user's current holdings across markets, GET /trades for historical trades, and GET /activity for actions like liquidity provision or redemptions. These endpoints often require specifying a user's address or an API key, as they may return sensitive account information. For example, a portfolio tracking app can use the Data API to retrieve all open positions of a given wallet (how many Yes or No shares the user holds in each market) and their current valuations. Similarly, traders can query their past trades or overall PnL. The Data API complements the Gamma and CLOB APIs by providing the personalized context that those general endpoints do not cover.

- **WebSocket Feeds (Real-Time Updates)** — In addition to REST endpoints, Polymarket offers WebSocket channels for live data streaming. The primary WebSocket service is at wss://ws-subscriptions-clob.polymarket.com/ws/, with separate channels for **market updates** and **user-specific updates***[22]*. The market channel broadcasts real-time order book changes, price updates, and last trade prices for subscribed markets*[23][24]*. This allows developers to maintain an up-to-date order book or ticker without continuous polling. The user channel (which requires auth) pushes notifications about the user's own orders (fills, cancellations)*[25][26]*. Additionally, a separate Real-Time Data Stream (RTDS) WebSocket (wss://ws-live-data.polymarket.com) provides live feeds of external data integrated into Polymarket, such as live cryptocurrency prices and market comment streams*[27][28]*. These websocket feeds ensure low-latency access (~100ms) to crucial data for market-making algorithms*[29]*, whereas the REST APIs (Gamma, CLOB, Data) are suitable for on-demand queries and have slightly higher latency (~1 second for Gamma data indexing) *[30]*.

- **Subgraph and GraphQL (Alternative Access)** — Although not the primary focus of this paper, it is worth noting that Polymarket's on-chain data is also accessible via The Graph's subgraphs and Bitquery's GraphQL APIs*[12][31]*. Polymarket maintains open-source subgraphs that can be queried for on-chain events like trades, positions, and resolutions*[32]*. This provides a trustless way to verify data or gather historical information. The official Polymarket API, however, often draws from these sources and provides a more convenient REST interface with additional off-chain metadata. Developers can choose the subgraph for custom

queries or use the Polymarket REST API for simplicity. In this paper, we focus on the REST and WebSocket interfaces that constitute the Polymarket API product.

Table 1 below summarizes the Polymarket API's main components, their base URLs, and typical uses:

API Component	Base URL or Endpoint	Purpose
Gamma API (Market Data)	`https://gamma-api.polymarket.com`	Market discovery and metadata (events, markets, etc.)[17]. Provides titles, categories, volume, liquidity, outcome info, etc. *(Read-only, public)*.
CLOB API (Trading)	`https://clob.polymarket.com`	Order book and price data, and order submission[17]. Used for fetching current prices (`/price`, `/book`, etc.) and placing/canceling orders. *(Read-write for authenticated users.)*
Data API (User Data)	`https://data-api.polymarket.com`	User positions, trade history, and activity data[20]. Allows tracking of holdings and past trades by address. *(Read-mostly, some endpoints may require auth.)*
WebSocket (Market)	`wss://ws-subscriptions-clob.polymarket.com/ws/market`	Real-time public feed of order book updates and price changes[23]. Subscribe to specific token IDs to receive live bid/ask updates and last trade prices.
WebSocket (User)	`wss://ws-subscriptions-clob.polymarket.com/ws/user`	Real-time private feed for user's own order events[25]. Requires API key auth; sends live notifications of the user's order fills, cancellations, etc.
RTDS (Live Data Stream)	`wss://ws-live-data.polymarket.com`	Live external data feeds integrated with Polymarket[28]. Provides streaming crypto price quotes and market comments in real time (used for UI and advanced tools).

Table 1: Polymarket API Components and Endpoints

This architectural separation improves both security and performance. For example, the trading API (CLOB) can be tightly rate-limited and secured since it

deals with state changes and user funds, while the Gamma API can be globally accessible and cached for heavy read traffic (market exploration by many users). The Data API isolates personal data queries. Meanwhile, real-time needs are served by WebSockets to avoid polling. In practice, a developer building on Polymarket will often use a combination of these: e.g., use Gamma API to get a list of interesting markets and their basic info, then use the CLOB API (or WebSocket) to get live prices for those markets, and possibly use the Data API to track user portfolios or performance.

In the next section, we present a detailed documentation-style breakdown of the key endpoints and data schemas provided by the Polymarket API, focusing primarily on the REST endpoints of the Gamma and CLOB APIs (as these cover most of the "every endpoint and field" requirement). We will then illustrate how these endpoints can be used in practice through example scenarios.

Detailed API Endpoints and Field Descriptions

In this section, we enumerate the major endpoints of the Polymarket API and describe the data structures (fields) they return. We follow a formal API documentation structure: for each endpoint, we note the HTTP method, path, a description of its purpose, required or optional parameters, and the format of the response including definitions of each field. To keep the presentation clear, we group endpoints by their functional category (Gamma API for market metadata, CLOB API for pricing/order book, Data API for user data). Key data objects (such as **Market**, **Event**, **Order Book**, etc.) are explained with their constituent fields and data types. Where applicable, we indicate how fields relate to underlying blockchain concepts.

Gamma API — Market Data Endpoints

The Gamma API is primarily used to retrieve information about events and markets on Polymarket. It organizes data in a hierarchy of **Series**, **Events**, and **Markets**, reflecting how markets are grouped:

· A **Series** is a collection of related events (for example, a sports league could be a series containing events for each game)*[33][34]*.

· An **Event** represents a real-world event or question being predicted, and it can contain one or multiple markets*[35]*. If an event has multiple mutually exclusive outcomes, Polymarket typically creates one market per outcome under that event

(this is referred to as a GMP — group of markets for a multi-outcome event, versus SMP — single market prediction for a binary event)*[36]*.

· A **Market** is the fundamental trading unit — essentially a specific outcome that people can buy or sell shares in*[35]*. Each market is tied to a unique on-chain condition (identified by a conditionId) and has an associated pair of outcome tokens (e.g., Yes and No tokens for binary markets) identified by internal clobTokenIds used in the order book system*[35][16]*.

Key Gamma API endpoints include listing all events or markets, retrieving details for a specific event or series, and searching. Below we document the most commonly used endpoints:

- **GET /markets — List Markets:** Retrieves a list of all markets, with various filtering and sorting options*[37][38]*. This is useful to get all individual markets (each corresponding to a single outcome) in one call. Supports query parameters like limit & offset for pagination, order (with ascending flag) to sort by fields (e.g., sort by volume or creation date), and filters such as tag_id (to filter by topic tag), closed (true/false to filter by market status), or condition_ids to fetch specific markets by their on-chain IDs*[39][40]*. The response is an array of market objects, each containing detailed fields listed in **Table 2.**

- **GET /events — List Events:** Retrieves a list of all events (groupings of one or more markets)*[41][42]*. Supports similar query parameters (pagination, sorting, tag filtering, closed status, etc.) as /markets for events*[43]*. Each event in the response includes high-level information about the event (title, description, category), as well as nested lists of its constituent markets (with their own fields). This endpoint is handy for getting an overview of each question and the outcomes offered. For example, an event “Who will win the 2024 election?” might contain markets for each candidate.

- **GET /markets/{id} — Get Market by ID:** Retrieves details for a specific market (by its id or slug). The response is a single market object (same structure as in the list) but may include additional context, such as the event it belongs to. This is used when you know a specific market’s identifier and want all details about it without pulling the entire list. (The documentation often handles this via the /events endpoint including markets; a direct /markets/{id} may be available similarly to the series endpoint pattern.)

- **GET /events/{id} — Get Event by ID:** Retrieves details for a specific event, including all markets under that event*[44]*. The response includes the event's metadata and a list of market objects (as in the list events call, but just for this event). Developers use this to get all outcomes of a multi-outcome event in one call. For a binary event, this would return the one associated market.

- **GET /series and GET /series/{id} — Series Listing:** These endpoints list series or fetch a specific series, which are higher-level groupings above events*[45][46]*. For example, a "Sports: NFL Season 2025" series might contain multiple events (games) which in turn have markets. The series object includes fields like title, seriesType, recurrence, and an array of events belonging to that series*[47][48]*. Series are useful for navigating hierarchies (especially in sports or recurring events).

- **GET /tags and GET /sports (Gamma Status):** The Gamma API also offers endpoints to retrieve available tags (categories) and sports leagues. GET /tags returns a list of tags (with fields like id, label, slug) that categorize markets (e.g., "Politics", "Crypto", "Sports")*[49][50]*. GET /sports may list configured sports leagues or groupings. Additionally, a Gamma Status endpoint likely provides a health check or version info for the Gamma service (not heavily used by clients except for debugging).

- **GET /search — Search:** A search endpoint allows querying across markets, events, and profiles by a keyword*[51][52]*. The result contains sections for matching markets, events, or user profiles. This is useful for building a search box where a user can type "president" and find all related markets/events.

Each of these endpoints returns JSON data with a rich set of fields. We now describe the **Market** and **Event** objects that appear in Gamma API responses. **Table 2** shows key fields for a Market (as returned by /markets or inside an event), and **Table 3** shows fields for an Event. Many fields are optional or may be null for certain types of markets (for instance, sports-specific fields appear only for sports markets). We focus on the most relevant fields:

Table 2: Market Object Fields (Gamma API)

Field	Type	Description
`id`	string	Unique market identifier (internal UUID or string) for this market. This is an identifier used by Polymarket's database and API.
`question`	string or null	The text of the market's question or outcome. For binary markets, this may be phrased as a yes/no proposition (e.g., "Will X happen by Y?"). For multi-outcome events, it might be a specific outcome statement (e.g., "Candidate A wins").
`conditionId`	string	The on-chain condition ID corresponding to this market's outcome set[53]. This 32-byte hex string (often 0x-prefixed) uniquely ties the market to the Conditional Tokens framework and is used in the smart contracts (e.g., to identify outcomes for resolution and token redemption)[16].
`slug`	string or null	URL-friendly slug for the market, often based on the question (e.g., `"will-x-happen-2025"`). Can be used in front-end URLs or as an alternate identifier for some API calls.
`category`	string or null	High-level category of the market (e.g., `"Politics"`, `"Crypto"`, `"Sports"`). Categories help group markets by subject.
`subcategory`	string or null	Sub-category if applicable (e.g., under "Sports", subcategory might be "Football"). Not always present.
`startDate`	string (date-time) or	Start date/time of the event or when trading opened, in ISO 8601 format (UTC)[54]. Some markets have a specified event start (e.g., a sports match kickoff

Field	Type	Description
	null	time).
endDate	string (date-time) or null	End date/time of the market (typically the deadline for outcome determination) in ISO format[55]. After this time, trading stops and the market awaits resolution.
active	boolean	Indicates if the market is currently active/open for trading[56]. true means trading is open; false means it may be paused or ended. (Contrast with closed.)
closed	boolean	Indicates if the market is closed (trading ended)[56]. Once closed is true, no new trades occur (market either awaiting resolution or already resolved). A market could be closed=true even if not yet resolved, if the trading period ended.
archived	boolean	Whether the market is archived/hidden from default listings[57]. Archived markets might be old or disallowed markets not shown in UI.
new	boolean	Marks if the market is newly created (recently launched)[58]. This can be used to highlight new markets.
featured	boolean	Marks if the market is featured/promoted by Polymarket[59]. Featured markets may get special visibility.
restricted	boolean	Indicates if the market has geographic or user restrictions[60]. Polymarket may restrict certain markets in certain jurisdictions.
resolutionSource	string or null	Text describing where the outcome will be determined from[55] (e.g., "Resolved via Official Election Commission results"). This guides users on what source the outcome will use.
resolvedBy	string or null	Identifier for who/what will resolve the market. Often "UMA" or a reference to the oracle adapter, since Polymarket uses UMA's oracle[5]. Could be null if not yet known.
umaResolutionStatus	string or null	Status of UMA oracle resolution (if using UMA), e.g., "PROPOSED", "SETTLED", etc., indicating whether the outcome is finalized on-chain. Null or empty if not applicable or not started.
questionID	string or null	The UMA question ID for the market[61]. If Polymarket used UMA's Optimistic Oracle, this is the unique ID of the price request on UMA's side. Useful for cross-

Field	Type	Description
		referencing UMA's system.
`marketMakerAddress`	string	The Ethereum contract address for the market maker or liquidity pool associated with this market[62]. In Polymarket's current implementation, this may refer to the specific contract handling the conditional token for this market (e.g., an FPMM in legacy markets or a proxy). This address holds collateral and interacts with the oracle.
`fee`	string or null	The market's fee (as a fraction or percentage) if applicable[63]. For example, Polymarket might have a trading fee or liquidity provider fee; this field might indicate that (in basis points or as a decimal).
`ammType`	string or null	The type of automated market maker, if any, supporting the market[64]. In older markets, this might be `"FPMM"` (Fixed Product Market Maker) or similar. In current order-book markets, this might be null or `"CLOB"`. The presence of `fpmmLive` (see below) also indicates if an AMM is active.
`fpmmLive`	boolean or null	Indicates if a fixed-product market maker (AMM) is currently active for this market[65]. In many markets now, this is `false` (since CLOB is primary). If `true`, it means an on-chain AMM is providing liquidity in parallel (this could occur for some legacy markets or special cases).
`enableOrderBook`	boolean	Indicates if the order book (CLOB trading) is enabled for this market[66]. Newer markets have `enableOrderBook=true`. If false, it might be an old market that was AMM-only.
`denominationToken`	string or null	The collateral token used (likely USDC) in this market[63]. Usually an address or symbol of the token that denominates the bets. Polymarket traditionally uses USDC (which on Polygon has a known address).
`outcomes`	string or null	Outcome labels or metadata[67]. Could be a JSON string or comma-separated names of outcomes. For binary markets this might be `"Yes,No"`. For multi-outcome markets, a list of outcome names.
`outcomePrices`	string or null	Current prices for each outcome[67]. Often, Polymarket represents this as a JSON array (in string form) or a comma-separated list of probabilities for outcomes. In a binary market, this might be a single probability (for "Yes") since "No" would be 1 minus

Field	Type	Description
		that. For multi-outcome, it could list each outcome's implied probability (they should sum to ~1). These prices are derived from either the AMM or the best bids/asks on the order book.
`shortOutcomes`	string or null	Possibly abbreviated outcome labels[68] (e.g., "Yes/No" or team abbreviations).
`volume`	string or null	Total trading volume of this market (usually in USDC) as a string[69]. This is cumulative volume traded since market inception. It may include both AMM and CLOB volume.
`volume24hr`	number or null	24-hour trading volume for this market[70]. Similar fields exist for 1 week, 1 month, 1 year (`volume1wk`, `volume1mo`, `volume1yr`)[70]. These are useful for gauging recent activity. They are typically numeric (could be in USDC).
`volumeAmm`	number or null	Total volume executed via the AMM (if any)[71]. Likewise `volumeClob` for volume via the order book[71]. These fields break down the volume by trading mechanism. In fully order-book markets, `volumeClob` ~=`volume`, and `volumeAmm` might be 0.
`liquidity`	string or null	Current liquidity pool size or metric[72]. In AMM context, this often meant the total amount of collateral in the pool. In the order book context, Polymarket defines a liquidity metric (perhaps the sum of open interest or some depth measure). Often this is given as a string representing a currency amount of how much is at stake. Market makers look at this to understand market depth.
`liquidityAmm`	number or null	Liquidity provided by AMM (if applicable)[73].
`liquidityClob`	number or null	Liquidity available in the order book[73]. These could be specific metrics (like sum of top X bids and asks). Polymarket documentation doesn't fully detail the calculation, but it is a comparative figure.
`openInterest`	number or null	Total open interest in the market (usually measured as total shares or total collateral at stake)[74]. This reflects how much value is currently invested in the market (distinct from volume, which is cumulative trading). For binary markets, open interest can be the sum of Yes and No shares that are currently in

Field	Type	Description
		circulation (should equal collateral locked).
`bestBid`	number or null	The highest bid price currently on the order book for this outcome token[75]. For a Yes outcome token, this is the highest price someone is willing to pay (in USDC) for a Yes share. Represented typically as a probability (0 to 1) or a price in currency (Polymarket normalizes so 1.00 = 100% = 1 USDC for a yes if it wins).
`bestAsk`	number or null	The lowest ask price on the order book[76]. This is the cheapest price at which someone is offering to sell a share. If `bestBid` is 0.40 and `bestAsk` is 0.45, it implies the market's current odds are in that range (40%-45% for Yes, in a binary market example).
`lastTradePrice`	number or null	The price at which the most recent trade was executed for this market[77]. This gives a sense of the latest market price. It may lie between the best bid and ask if a trade just crossed the spread, or equal to one of them if it matched that price.
`oneDayPriceChange`	number or null	The percentage or absolute change in price over the last 24 hours[78]. Similar fields exist for 1 hour, 1 week, etc. (e.g., `oneHourPriceChange`, `oneWeekPriceChange`)[78][79]. These indicate how much the odds have moved, which is important in volatile markets. Likely measured in percentage points or relative change from previous day's price.
`makerBaseFee`	integer or null	The base fee (in basis points) for makers in this market[80]. Polymarket introduced a fee/rebate structure where takers pay fees and makers might earn rebates[81]. A nonzero `makerBaseFee` might indicate a negative fee (rebate) if Polymarket rewards liquidity providers, or 0 if makers don't pay fees.
`takerBaseFee`	integer or null	The base fee (bps) for takers in this market[80]. For example, `takerBaseFee: 156` could mean a 1.56% taker fee at 50% odds, as indicated by Polymarket's dynamic fee schedule[81]. Fees can vary by market type (some markets like crypto 15-minute markets have taker fees up to 1.56%[81]). This field informs algorithmic traders of the fee overhead.
`negRisk`	boolean or null	A flag indicating if the market is a **negative risk** market (used in multi-outcome scenarios)[82]. Negative risk markets allow combined positions that guarantee a profit (due to overlapping outcomes); Polymarket flags

Field	Type	Description
		such markets and sometimes applies special rules or fees (`negRiskFeeBips`). If `negRisk=true`, market makers might be cautious as certain arbitrage opportunities exist by buying all outcomes.
[Various image fields]	string or object	The market object includes fields for images and icons (e.g., `image`, `icon`, and their optimized variants) which contain URLs to illustrative graphics for the market or event[83][84]. These are mainly for front-end use. Each has an `Optimized` sub-object with CDN links and sizes. We omit detailed description as they do not affect trading logic.
`tags`	array of objects	A list of tag objects associated with the market[85]. Each tag object has an `id`, `label` (name of the tag), and `slug`. Tags classify the market (e.g., "Sports", "US Politics", "Crypto"). This duplicates some info from category but is more granular.
`categories`	array of objects	Similar to tags, but a hierarchy: category objects may include a `parentCategory` or grouping[86]. Often one primary category and possibly subcategory is listed. This helps in filtering markets by broad categories.
`series`	array of objects	If this market is part of a series, the series it belongs to (and potentially related series) can be listed here[87]. Usually an event links to a series, and the market links to the event.
`events`	array of objects	The event(s) this market is associated with[88]. Typically a market has exactly one parent event. This field might appear with the event's basic info (id, slug, title) here for reference.

Table 2: Key fields of a Market object returned by Gamma API. (Note: The table focuses on important fields. The Polymarket API returns many additional fields, but not all are critical for understanding usage; some are for internal or UI purposes. Fields like createdBy, updatedAt, etc., which track creation metadata, are omitted here for brevity.)

Most of these fields in Table 2 are visible in the JSON structure of a /markets or /events response*[89][90]*. As an example, a portion of a single market JSON might look like:

{
“id”: “abc123”,
“question”: “Will Alice win the election?”,
“conditionId”: “0x1234…abcd”,
“slug”: “will-alice-win-election”,

“category”: “Politics”,
“startDate”: “2025–11–01T00:00:00Z”,
“endDate”: “2025–11–08T00:00:00Z”,
“active”: true,
“closed”: false,
“resolutionSource”: “Official electoral commission results”,
“outcomes”: “[\”Yes\”,\”No\”]”,
“outcomePrices”: “[0.65, 0.35]”,
“volume”: “150000.00”,
“liquidity”: “50000.00”,
“openInterest”: 100000,
“bestBid”: 0.64,
“bestAsk”: 0.66,
“lastTradePrice”: 0.65,
“oneDayPriceChange”: 0.05,
“makerBaseFee”: 0,
“takerBaseFee”: 100,
“negRisk”: false,
“tags”: [{ “id”: “1”, “label”: “Politics”, “slug”: “politics” }],
…
}

This example indicates a Politics market where Alice is currently favored (Yes shares at 0.65 or 65%). Volume and liquidity are given in USD terms (strings for precision). The best bid/ask spread is 0.64–0.66. In the last day, the price moved by +0.05 (maybe from 0.60 to 0.65). Maker fees are zero, taker fee is 1% (100 bps). The market is active and not yet closed.

Next, **Table 3** outlines the Event object fields. Many fields overlap with Market (since an event aggregates markets), but events have their own properties:

Table 3: Event Object Fields (Gamma API)

Field	Type	Description
id	string	Unique identifier of the event (could be a UUID or numeric string).
title	string	Title of the event (a descriptive name of the underlying question)[91]. For example: "2025 City Mayor Election".
subtitle	string or null	Subtitle or additional context for the event (if provided). Possibly used for clarifying multi-part questions or series info.
description	string or null	A longer description of the event/question. Might include rules or details on how the event will be judged.
resolutionSource	string or null	A description of where the resolution data will come from, similar to market's field[92]. At event level, if multiple markets share the same resolution criteria, it can be here.
image/icon	string or null	URLs to an image or icon representing the event (for UI purposes).
category	string or null	Category of the event (e.g., "Politics"). Typically events carry a category which is inherited by its markets.
startDate	string (date-time) or null	Start time of the event (if applicable)[93]. For sports events, this is game start; for other events, it might not be used.
endDate	string (date-time) or null	End time or deadline of the event (after which outcome is known)[94]. All markets in the event likely share this.
active	boolean	Whether at least one market in the event is active. Often reflects if event's markets are tradable at the moment[56].
closed	boolean	Whether the event's markets are all closed[56]. True usually when the event time passed and trading stopped.
archived	boolean	If the event (and its markets) is archived/hidden[57].
featured	boolean	If the event is featured/promoted.
restricted	boolean	If the event is restricted by geolocation or other criteria.
liquidity	number	The aggregate liquidity of all markets in the event

Field	Type	Description
		(perhaps sum or weighted; often equal to the main market's liquidity if one market, or sum if multiple)[74].
`volume`	number	The aggregate volume of all markets in the event[74]. Useful for multi-outcome events (sum of volumes) or equal to single market volume.
`openInterest`	number	The total open interest across the event[74]. In multi-outcome events, could be sum of open interest of each outcome, which might double-count collateral unless outcomes are exclusive. Polymarket might compute event openInterest differently (e.g., maximum collateral at stake among outcomes).
`volume24hr` (etc.)	number	24h (and 7d, 1mo, 1yr) volume for the event[70]. These aggregate the volumes of the event's markets over those periods.
`commentsEnabled`	boolean	If commenting (discussion) is enabled for this event[95]. Polymarket allows users to discuss markets; this flag shows if that feature is on.
`negRisk`	boolean	If **negative risk** is enabled at the event level[96]. For multi-market events, Polymarket sometimes enables an automatic mechanism to prevent sure-win arbitrage. If true, there might be a special "Other" market or a fee to deter arbitrage. For example, in an event with multiple markets covering all outcomes, `negRisk=true` signals that owning a complete set of outcome shares yields a risk-free payoff (so the market maker might charge a fee to discourage exploiting that).
`negRiskMarketID`	string or null	The market ID of any special negative-risk market (like "Another outcome" market)[96]. If the event has an "Other" outcome to complete the outcome space, its market ID could be referenced here.
`negRiskFeeBips`	number or null	If negative risk is true, this is the fee (in basis points) applied to eliminate arbitrage[96]. E.g., 50 means a 0.5% fee for trading combinations of outcomes.
`markets`	array of objects	**List of markets under this event**[97]. Each market object here has the same structure as in Table 2 (often a slightly abridged version in the nested context). This is the key part of the event: it enumerates each outcome market with its

Field	Type	Description
		question, prices, etc. For single-outcome events, this list has one entry. For multi-outcome events, multiple markets are listed.
`series`	array of objects	The series this event belongs to (if any)[98]. Usually one series object with fields like series `title`, etc. Helps to navigate up the hierarchy.
`parentEvent`	string or null	If this event is a sub-event (e.g., part of a bigger event), the parent event's ID might be here[99]. (Polymarket supports event hierarchies, though rarely used.)
`eventCreators`	array of objects	The creators or authors of this event (community created markets often credit a user)[100]. Each object may have an `id`, `creatorName`, handle, URL, etc. Not crucial for API usage, but provides attribution.
`tweetCount`	integer	If Polymarket tracks Twitter activity related to this event (in some cases they do for trending), number of tweets. This is a niche field.
`chats`	array of objects	If the event has associated chat channels (possibly for live discussion), they might be listed[101]. Each with `channelId`, name, etc. (This is more UI/social data.)
`live`	boolean	For sports events, indicates if the game is currently live (in-progress)[102]. Polymarket can update this for live sports markets.
`elapsed`, `period`, `score`	various types	Sports-specific fields: e.g., `score` might hold current score, `period` the quarter/half, `elapsed` time elapsed in game[103][102]. These are updated for live sports to drive in-play markets.
`gameStatus`	string or null	Another sports field indicating game state (e.g., "Scheduled", "Final")[104].
`cyom`	boolean	"Create Your Own Market" flag (if this event was user-created through Polymarket's builder program)[105]. CYOM events might have different moderation.
`automaticallyResolved`	boolean	Indicates if the event will be auto-resolved via oracle (true for UMA-based markets)[106]. If false, perhaps manual resolution or not yet integrated. Most markets should be true (Polymarket uses the oracle).

Field	Type	Description
`automaticallyActive`	boolean	Indicates if the markets go active automatically at a certain time (some events might schedule market opening)[106].
`enableOrderBook`	boolean	If order book trading is enabled for all markets in this event[107]. Usually follows market-level flags; if an event was created when orderbooks were off, this could be false.
Other fields...	*various*	The Event object can include many other fields (e.g., template info for templated markets, references to related data, etc. as seen in the documentation snippet). These are beyond the scope of normal API usage. For instance, `templates` (list of template definitions if event was created from a template), `pendingDeployment` flags if markets are scheduled but not yet on-chain, etc.[108][109]. These are mostly relevant to Polymarket's internal market creation system and can be ignored by most API consumers.

Table 3: Key fields of an Event object returned by Gamma API. (Again, focusing on commonly used fields. Events primarily serve to group markets, so the most important part of an Event object for developers is typically the list of markets with their prices and IDs.)

From these descriptions, developers can understand that to get the current odds of an event, one might either call /markets (to directly get each market's outcome price) or call /events to get events which include their markets. For instance, if one wants the probabilities of each candidate winning an election (a multi-market event), the /events/{id} call will return all outcome markets with their outcomePrices or best bids/asks.

It's worth noting the **relationship between prices and probabilities**: In Polymarket, outcome token prices are essentially probabilities (when expressed in USD for a $1 payout). A price of 0.65 for "Yes" means the market believes there's a 65% chance of "Yes". If multiple outcomes exist, buying one outcome token is like betting on that outcome; complete sets of outcome tokens can be redeemed for the collateral regardless of outcome (ensuring no-arbitrage if prices sum to 1 after fees). The API often provides outcomePrices as a convenient snapshot of the market's implied probabilities*[72]*.

CLOB API — Price and Order Book Endpoints

The CLOB (Central Limit Order Book) API deals with real-time trading data. It exposes endpoints for retrieving current prices, order book snapshots, and historical price information. Some endpoints also allow submitting orders (with authentication). We document the key read-only endpoints here, as they are essential for most integrations (trading bots, data analysis):

- **GET /price — Get current price for a token:** Returns the current market price for a given outcome token, on a specified side (buy or sell)*[110][111]*. Required query parameters: token_id (the identifier of the outcome token in the order book) and side (either BUY or SELL). The Polymarket system assigns each outcome token (e.g., Yes or No of a market) a unique token_id, which is typically obtainable from the market data (in the clobTokenIds field*[112]*). If side=BUY, the API returns the lowest ask price (what you'd pay to buy one token); if side=SELL, it returns the highest bid (what you'd get for selling one token). The response is a simple JSON with a "price" field as a string*[113][114]*. For example, a query to /price?token_id=123456&side=BUY might return { "price": "0.6500" }, meaning you can buy that outcome at 0.65. This is essentially the top-of-book quote.

- **GET /book — Get order book for a token:** Retrieves the current order book for the specified token (outcome) in the market*[115]*. The query parameter is typically token_id. The response includes the aggregated buy and sell orders (bids and asks) at various price levels. It usually has a structure like { "bids": [{ "price": ..., "size": ...}, ...], "asks": [{ "price": ..., "size": ...}, ...], "market": { ... } }. The market sub-object may include metadata about the market (such as min order size, tick size, etc., as Polymarket added in an update*[15]*). This endpoint allows an application to see the depth of the market: how much liquidity is available at each price. For example, one could iterate through bids to calculate total buy-side liquidity at or above certain price. **Note:** Polymarket also provides GET /books (plural) to fetch multiple order books in one call, and a batch request format for similar bulk queries*[116]*, which can be useful for efficiency.

- **GET /midpoint — Get midpoint price:** Returns the midpoint between the best bid and best ask for a given token*[117]*. Query param: token_id. The midpoint is often used as a reference "market price" especially if the spread is tight. E.g., if best bid is 0.64 and ask is 0.66, midpoint is 0.65. The response JSON might be { "midpoint": "0.6500" }. This value can be seen as the current fair price estimate.

- **GET /price-history (or /prices-history) — Get price history for a token:** Returns historical price data for a given outcome token*[118][119]*. Query params include either an interval (like 1d for last 1 day, 1h, 1w, etc.) or a startTs & endTs for a custom range, plus an optional fidelity (granularity in minutes)*[120]*. The response is typically an array of data points, each with a timestamp and price, often under a field like "history": [{"t": 1630454400, "p": "0.52"}, …]. This is useful for charting or analyzing trends. For example, one can fetch a market's hourly price history over the last week to see how the probability evolved. This data is derived from trade logs (order fills) and perhaps interpolated at regular intervals.

- **GET /last-trade-price — Get last trade price:** Returns the last traded price for the token*[121]*. This might be similar to the lastTradePrice field in Gamma, but retrieved directly from the CLOB data. It's useful if you want confirmation of the last execution separate from midpoint.

- **GET /spread — Get bid-ask spread:** Returns the current spread for the token*[122]*. Likely gives the best bid and ask together (as an object with bid and ask prices)*[123]*. This is essentially the same info as asking for /price on both sides, but combined.

- **GET /tick-size — Get minimum tick size:** Returns the minimum price increment (tick size) for the token*[124]*. Polymarket might allow different tick sizes for different markets (e.g., $0.001 increments). The response gives tickSize as a string or number representing that smallest step*[125]*. This is important for algorithmic traders to know how to place orders.

In addition to these read endpoints, the CLOB API includes **authenticated endpoints** for order management:

- **POST /order — Place an order:** Allows placing a limit order. The payload must include order details like token_id, side (buy/sell), price, size, and usually a user apiKey or signature for auth. Given Polymarket's design, the client actually needs to generate an EIP-712 signature for the order and include it (or use an API key if Polymarket's system allows placing on behalf of user). The specifics: Polymarket's API likely expects something like { "market": <conditionId>, "outcome": <0 or 1>, "type": "limit", "side": "buy", "price": "0.65", "size": "100", … signature … } or if using the builder API key, just the key and order details. The docs reference a **Relayer Client** and an API key system for builders*[126][127]*,

implying that to place orders programmatically, one must enroll as a Polymarket Builder, obtain keys, and then use their authenticated endpoints. Due to the scope of this paper, we won't delve deeply into the exact mechanics of signing orders, but it's notable that the API does not simply take plaintext orders for security — orders must be authorized.

- **DELETE /order — Cancel an order:** Cancels a specific order, usually by order ID or a combination of parameters identifying the order. Requires authentication (only the user who placed the order can cancel). After cancellation, the on-chain funds reserved for that order become available again to the user's balance (the API and exchange handle this via the proxy wallet mechanism; each user has a proxy wallet contract*[6]* that actually holds their funds on-chain).

- **Batch order endpoints:** Per the changelog, Polymarket introduced an endpoint to place multiple orders in one request (batch)*[128]*. This likely is POST /orders (plural) or a similar path, taking an array of order definitions. This helps market makers to update multiple levels of an order book efficiently. They increased the batch size from 5 to 15 orders per call in 2025*[129]*.

- **Heartbeats and status:** A mention in the changelog of a "HeartBeats API" for monitoring connection status*[130]* suggests there might be an endpoint to ping or keep-alive, possibly to use in long-running trading bots to ensure they are still connected (maybe relevant for WebSocket or order matching cancellations if a client goes offline). This is more of an advanced tool.

For **order and trade data**, while the CLOB API might not provide a direct GET endpoint for all past trades (to avoid heavy data pulling publicly), the Data API or subgraph is intended for that. However, one can subscribe to the WebSocket or use the subgraph for trade fills. The Data API's GET /trades can fetch historical trades either by user or market*[131][132]*. If called without a user, possibly it returns recent trades for a market (if allowed).

In sum, the CLOB API gives the tools needed to replicate a trading interface: you can fetch the current state (price and order book), watch for updates, and send orders. The Polymarket documentation emphasizes that these endpoints allow programmatic trading similar to a traditional exchange API*[133]*, with all the data needed to make decisions (prices, spreads, etc.) available in real time.

Data API — User Data Endpoints

The Data API is slightly more specialized and often requires API keys since it deals with private data. For completeness, we list its primary endpoints and their purpose:

- **GET /positions — Get user positions:** Returns the list of all holdings for a given user address*[134]*. The query parameter is usually user=<address>. Optionally, one can filter by market=<marketId> to get positions in a specific market. The response contains an array of position objects*[135]*. Each position object might include fields like marketId (or conditionId), outcome identifier, amount of shares held, average cost, unrealized P/L, etc. For instance, it may show that user 0xABC holds 100 Yes shares and 50 No shares in a certain market. This is extremely useful for portfolio tracking: a developer can use this to display a user's current bets and their values given latest prices (value calculation might not be directly given, but one can multiply shares by price from Gamma/CLOB data).

- **GET /trades — Get trade history:** Returns a list of trades, either for a specific user or market*[136]*. It accepts optional user=<address> and/or market=<marketId> and supports pagination (limit, offset)*[137]*. The output is an array of trade objects*[138]*. Each trade object likely includes details such as timestamp, market, outcome, size, price, side (buy/sell and whether user was taker or maker), fee paid, and maybe an order ID reference. This is essentially a transaction history. If no user is specified, Polymarket might allow retrieving recent trades in a market (public data), but for user-specific filtering, authentication might be needed to get more than what's public. (However, since all trades are on-chain events, even user trades are technically public, but the API could offer more convenient filtering.)

- **GET /activity — Get user activity:** Combines various user actions (bets, liquidity adds, redemptions) into a timeline. Each activity entry might be a trade, an order placed or canceled, a market resolved (and winnings collected) etc. This endpoint provides a one-stop history log. It likely supports user param and pagination.

- **GET /holders — Get market holders:** This endpoint (which we saw in the documentation snippet) provides the top holders of outcome tokens for a given market*[139][140]*. It requires a query param market=<conditionId> (and is capable of multiple markets at once by specifying multiple IDs). The response lists, for each outcome token of the market, the top addresses (or proxy wallet

IDs) holding that token and how much*[141][142]*. For example, in a binary market, you'd get two lists (Yes token holders and No token holders) with the largest positions. Each holder entry can include the holder's proxy wallet address, a username if known (Polymarket has profiles), and the amount held*[141]*. This is useful for analysis of market distribution (e.g., how concentrated the holdings are). It's a read-only endpoint that doesn't require user auth since it's aggregate public info, but it's under Data API as it's not core market metadata.

- **GET /builder/profile (or similar under /builders):** Polymarket Builders Program endpoints might exist to retrieve information about API keys and usage (for developers enrolled in their program)*[126]*. These are administrative and not related to market data.

- **Authentication for Data API:** Some Data API calls may require an API key or token in the header (especially if retrieving data tied to a specific user). Polymarket likely uses API keys linked to user accounts for high-privilege queries. The "Builder Profile & Keys" section*[143]* indicates that developers can generate keys to use these APIs. In our context, we assume read access to public data (like market holders or public trades) is allowed without auth, but private queries (like a random address's positions) might be restricted to the owner for privacy unless the user's data is public. Given blockchain data is public, Polymarket might not restrict reading positions by address, but the presence of usernames and profiles in the output suggests some data is user-opt-in (e.g., pseudonym).

Underlying Smart Contract Interaction

It's important to note how the above endpoints correlate with on-chain activity: — **Market creation** (reflected in Gamma API: new events/markets) corresponds to on-chain events like ConditionPreparation in the Polymarket main contract*[13]*. The Gamma API uses off-chain indexing to immediately list new markets with metadata (title, etc.) that might have been embedded in the UMA oracle question's ancillary data. — **Price and Order Book** (CLOB API) correlate to the state of the on-chain exchange contract. When orders are matched via the API, the exchange contract emits OrderFilled and OrderMatched events*[144]*. Polymarket's systems update the order book and the Gamma API's lastTradePrice etc. accordingly. The price history endpoint likely compiles data from these trade events. — **Positions** (Data API) correspond to the balances of outcome tokens in user proxy wallets. Every user

trade results in transfers of ERC-1155 outcome tokens between wallets (the exchange contract to user's proxy or vice versa). The Data API likely reads from a subgraph or directly from the blockchain to calculate current balances*[145]*. The open interest field in markets is essentially a measure of total token supply (which equals total collateral locked in the conditional token contracts). — **Resolution** status (fields like umaResolutionStatus) reflect the state of a request to the UMA Optimistic Oracle*[5]*. When a market's outcome is decided, the Polymarket UmaCtfAdapter contract will have its settle() function called after UMA resolution, and the market's conditional tokens become redeemable. The API might then update closed=true and eventually reflect resolution by marking the market as resolved/settled (possibly by removing it from active lists and enabling a "redeem" action via their UI or API).

By providing this API abstraction, Polymarket ensures developers do not need to manually handle these blockchain interactions — such as tracking events or calling contracts to get balances. Instead, one can trust the API to present a coherent view of the state. However, advanced users can cross-verify with Polymarket's open subgraph (GraphQL) for full decentralization assurance*[12]*.

Formal Field Reference Tables

For quick reference, we summarize some of the important fields introduced above, in a more formal documentation style:

- **Market fields and their meanings:** (See Table 2 for comprehensive list) Key ones include id, conditionId (on-chain ID)*[53]*, slug, question, category, time fields, status flags (active, closed etc.)*[56]*, financial stats like volume, liquidity*[72]*, and price points like bestBid, bestAsk, lastTradePrice*[77]*. These fields allow a developer to know everything about a market's current state and historical performance needed for decision-making.

- **Event fields:** (See Table 3) including title, description, aggregated volume and liquidity, and the list of child markets. The event acts as a container and is especially relevant if dealing with multi-outcome scenarios.

- **Order Book fields:** A single order entry might have price (in string to maintain precision, representing the probability/price) and size (quantity of outcome tokens at that price). The order book endpoint's response likely also has min_order_size, tick_size (market metadata Polymarket added*[82]*). For instance, Polymarket enforces a minimum order size (to prevent dust orders) which is given as min_order_size in the get-book response*[82]*. It also flags if a

market is neg_risk (so the API consumer can be aware)*[82]*. These metadata fields in the order book are clearly documented in the changelog entry.

- **Trade fields:** A trade object (from Data API or WebSocket) might include: timestamp, marketId, outcome (token) ID, size, price, makerOrderId, takerOrderId, and possibly side (Polymarket added a side field to indicate whether the maker was buy or sell in each trade*[146]*). This clarifies the nature of the trade, which is useful for reconstructing order flow.
- **Position fields:** A position from /positions might include marketId, outcomeIndex (e.g., 0 for No, 1 for Yes, or the index of outcome in multi-outcome), size (number of tokens held), and pendingReward or similar if a market resolved but not redeemed. It may also surface the user's proxyWallet address which actually holds those tokens on-chain.
- **Profile fields:** If using the search or holders endpoint, you might see user profile fields like name (username), pseudonym, profileImage. These are part of Polymarket's social layer. They don't affect trading but appear in data.

For brevity, we will not list every field from the documentation in tables (the Polymarket API returns over a hundred fields per market as seen in the docs*[147]* *[148]*), but we have covered the ones most pertinent to blockchain developers. The **Gamma API Overview** documentation itself highlights *Key Fields* for market makers: conditionId, clobTokenIds, outcomes, outcomePrices, volume, liquidity*[16]* as critical pieces of data, which aligns with our focus.

In the next section, we will demonstrate how these endpoints and fields come together in practical usage through example scenarios and code snippets.

Use Cases and Examples

In this section, we explore common use cases for the Polymarket API and illustrate how a developer might utilize the endpoints for each scenario. We cover four primary use cases as requested: **(1) Fetching markets**, **(2) Querying outcomes**, **(3) Interpreting liquidity**, and **(4) Making predictions**. Each sub-section will describe the goal and then provide a code example (with annotations) either in Python or JavaScript to demonstrate the API calls and data handling.

These examples assume no prior specialized Polymarket SDK — they use direct HTTP requests to the API, which is what an independent developer might do. (In practice, Polymarket also offers client libraries, e.g., a Python SDK for the CLOB and a TypeScript client for WebSockets, which abstract some of these details*[149][150]*. However, understanding the raw API usage is instructive.)

1. Fetching Market Listings

Use case: A developer or researcher wants to retrieve a list of active markets on Polymarket, perhaps to display them on a dashboard or to filter for markets of interest (for example, all active politics markets sorted by volume).

Using the Gamma API's GET /markets or GET /events endpoints, one can fetch these listings. For a list of individual markets with their prices, GET /markets?closed=false is a straightforward approach (it will return only markets that are not closed, i.e., currently trading). If the interest is in distinct questions, GET /events?closed=false would return each event (question) with its markets, which is better if one wants to avoid double-counting multi-outcome questions.

Example: Fetch all active markets and print their title (question), current Yes price, and 24h volume. We will use Python with the requests library for the demonstration:

```
import requests

# Define the Gamma API endpoint for markets
url = “https://gamma-api.polymarket.com/markets"

# Fetch only active (open) markets using query parameter
params = {“closed”: “false”, “limit”: 50} # limit to 50 for this example
response = requests.get(url, params=params)
markets = response.json() # parse JSON response into Python list/dict
```

```
print(f"Retrieved {len(markets)} markets.")
for m in markets:
title = m.get("question") or m.get("description") or "(No title)"
# outcomePrices is a string of list, need to parse it
price_info = m.get("outcomePrices")
current_price = None
if price_info:
try:
# outcomePrices might be a JSON string like "[0.65, 0.35]" for Yes/No
prices = eval(price_info) # or json.loads if it's a JSON array in string
# Assume outcome index 1 corresponds to "Yes" in binary markets:
current_price = prices[0] if len(prices) > 0 else None
except Exception:
current_price = price_info # if parsing fails, just use raw string
volume_24h = m.get("volume24hr", 0)

print(f"Market: {title[:50]:50} | Yes Price: {current_price} | 24h Volume:
{volume_24h}")
```

Annotated explanation: In the code above, we send a GET request to the /markets endpoint with closed=false to filter out closed markets. The limit parameter is set (the API defaults might return 20 or so if not specified, and can be increased up to some max like 50 or 100). The response is a JSON array of market objects. We iterate through each market (m) and extract a title (Polymarket sometimes uses question for binary markets and title for events; here each m is a market so question is appropriate if present). We also fetch the outcomePrices field; since it's given as a string in the JSON, we parse it to extract the numerical price for the primary outcome. We then print a line showing the market title, the price (assuming index 0 corresponds to Yes or the main outcome), and the 24-hour volume.

Sample output (illustrative):

```
Retrieved 47 markets.
Market: Will Alice win the election? | Yes Price: 0.65 | 24h Volume: 120000
Market: Will Bitcoin price exceed $50k on 2025–12–31? | Yes Price: 0.32 | 24h Volume:
450000
Market: Ethereum 15-minute price (Up vs Down) | Yes Price: 0.58 | 24h Volume:
```

```
80000
...
```

This demonstrates listing markets. A similar approach in JavaScript (using fetch) could be done in a Node or browser environment:

```
async function fetchMarkets() {
const res = await fetch("https://gamma-api.polymarket.com/markets?
closed=false&limit=20");
const markets = await res.json();
console.log(`Fetched ${markets.length} markets.`);
markets.forEach(m => {
const title = m.question || m.description || "(No title)";
let yesPrice = null;
if (m.outcomePrices) {
try {
const prices = JSON.parse(m.outcomePrices);
yesPrice = prices[0];
} catch {
yesPrice = m.outcomePrices;
}
}
console.log(`${title} — Yes Price: ${yesPrice}, 24h Vol: ${m.volume24hr}`);
});
}
```

This would achieve the same result in JS. Note how we had to parse the outcomePrices which came as a JSON string. (If Polymarket changes that to a structured array in future, the parsing would adjust.)

The developer can also use filters like tagId if they want only specific categories. For example, Polymarket's tags list might show that "Politics" has tag id 123; then adding ?tag_id=123 filters only politics markets*[151]*.

2. Querying Outcomes and Prices

Use case: Given a specific event or question, retrieve the outcomes and their current probabilities. This might be used in a bot that monitors a particular event's odds, or an application that displays detailed info for one market.

There are two main ways: — Use GET /events/{id} to get the event and its markets (with prices of each outcome). — Or if one knows the specific market IDs for outcomes, use /markets/{id} for each (or if they are known token IDs, directly use CLOB price endpoints).

The easier method is via Gamma: GET /events/{eventId} yields an event with a markets array. Each market in that array will have an outcomePrices or at least a bestBid/bestAsk from which we can derive the price.

Example: Query a specific event (by slug) and output each outcome's probability. Suppose the event slug is "2025-mayor-election".

```
import requests

event_slug = "2025-mayor-election"
url = f"https://gamma-api.polymarket.com/events/{event_slug}"
response = requests.get(url)
if response.status_code == 200:
event = response.json()
title = event.get("title")
print(f"Event: {title}")
markets = event.get("markets", [])
for market in markets:
outcome_label = market.get("question") or market.get("groupItemTitle") or
"Outcome"
# For outcomes, Polymarket might put the question or a group title.
price = None
# Check if outcomePrices is directly given (for markets with one outcome).
if market.get("outcomePrices"):
try:
prices = eval(market["outcomePrices"])
# If this market is binary, outcomePrices might be single value or [yes,no]
# If it's the only outcome (like an "Other"), treat as separate.
price = prices[0] if isinstance(prices, list) else float(market["outcomePrices"])
except:
price = market["outcomePrices"]
else:
# If no direct outcomePrices, use bestBid and bestAsk to estimate midpoint
```

```
probability
bid = market.get("bestBid")
ask = market.get("bestAsk")
if bid is not None and ask is not None:
price = (bid + ask) / 2 # midpoint estimate
elif bid is not None:
price = bid
elif ask is not None:
price = ask
if price is not None:
# Convert price to percentage probability
prob_pct = float(price) * 100
print(f" — {outcome_label}: {prob_pct:.1f}%")
else:
print(f" — {outcome_label}: price unavailable")
else:
print("Event not found or API error")
```

Explanation: Here we use the event slug to fetch the event. If the API allows direct slug access (as implied by Polymarket docs, slug can substitute ID*[152]*), we get the JSON for that event. We then loop through each market/outcome in the markets list. For each, we attempt to get a price. If outcomePrices is present, that directly provides the probability (for a binary market, it might be a two-element list [YesPrice, NoPrice]; we would take the relevant one. Typically each market is one outcome, so might just have a single price value in outcomePrices string, but Polymarket's schema might differ). If outcomePrices is not given (maybe it's given only at event level in some cases), we fall back to using bestBid/ask to compute a price (midpoint). Then print the outcome's label and probability.

In a multi-market event, market["question"] often holds the phrasing of that outcome (as shown in the Gamma Structure example: event "Where will X go to college?" with markets "Will X go to Y University?"*[153]*). If Polymarket used a field groupItemTitle for outcomes in a grouped event*[154]*, we attempt that too.

Output example:

```
Event: 2025 Mayor Election
— Alice (Candidate A): 63.5%
```

— Bob (Candidate B): 34.8%
— Any other candidate: 1.7%

This indicates the prices for each outcome. Here possibly three markets: Alice, Bob, Other. The probabilities sum ~100%. (The “Any other” outcome often covers the remainder in a negRisk scenario with a small probability and perhaps a fee.) The data suggests Alice is leading with ~63.5% implied probability.

For a binary event (single market), the event’s markets list has just one entry (the binary market). That market’s outcomePrices might be something like “[0.40, 0.60]” representing Yes=0.40, No=0.60. In such a case, our code would pick the first element for that market thinking it’s the outcome itself. We might need to refine logic to handle binary vs multi-outcome: — If event has one market with two outcomes, perhaps better to not use /events at all but use /markets directly or rely on the fact that outcomePrices as an array covers both. Alternatively, the event may itself list outcomes at top level, but likely not — the markets array is the place.

We could adjust: if only one market and it has multiple outcomePrices entries, iterate through both. But Polymarket doesn’t provide separate labels for yes/no in one market object; rather, in binary case, the market question itself implies yes or no. So it’s simpler to treat binary as one outcome “Yes” at price p, and infer “No” is 1-p.

Anyway, the approach stands: using event or market data to get probabilities.

In JavaScript, the same can be done:

```
async function printEventOutcomes(eventIdOrSlug) {
const res = await fetch(`https://gamma-
api.polymarket.com/events/${eventIdOrSlug}`);
if (!res.ok) {
console.error(“Failed to fetch event”);
return;
}
const event = await res.json();
console.log(`Event: ${event.title}`);
for (const market of event.markets) {
const label = market.question || market.groupItemTitle || “Outcome”;
let price = null;
```

```
if (market.outcomePrices) {
try {
const prices = JSON.parse(market.outcomePrices);
price = (Array.isArray(prices) ? prices[0] : prices);
} catch {
price = market.outcomePrices;
}
} else if (market.bestBid !== undefined || market.bestAsk !== undefined) {
const bid = parseFloat(market.bestBid);
const ask = parseFloat(market.bestAsk);
if (!isNaN(bid) && !isNaN(ask)) price = (bid + ask) / 2;
else if (!isNaN(bid)) price = bid;
else if (!isNaN(ask)) price = ask;
}
if (price != null) {
console.log(` — ${label}: ${(parseFloat(price)*100).toFixed(1)}%`);
} else {
console.log(` — ${label}: no price`);
}
}
}
```

This function will log each outcome.

This use case shows how to get outcome probabilities, which is a core reason to use Polymarket data (extracting the “wisdom of the crowd” prediction).

3. Interpreting Liquidity and Order Book Depth

Use case: A trader or market maker wants to gauge the liquidity of a market — how easy is it to execute a trade of a given size without moving the price? Or, for research, one might want to understand how much money is at stake at the current odds.

Polymarket provides several pieces of information related to liquidity: — The liquidity field (and liquidityNum, liquidityClob) in the market data, which gives a numeric measure of current liquidity*[72]*. — The full order book (via GET /book) to see granular depth (bids and asks and their sizes). — The openInterest field which tells total collateral at stake, an indirect measure of market depth.

To interpret liquidity, one often needs the order book. For example, if we want to know how much can be bought before the price goes up by X%, we would sum ask sizes up to a certain price level.

Example: Fetch the order book for a specific outcome token and calculate the total available volume within 10% of the current mid price on both sides.

```
import requests

token_id = “123456” # hypothetical token id for Yes outcome of a market
# Get order book for this token
res = requests.get(“https://clob.polymarket.com/book", params={“token_id”: token_id})
orderbook = res.json()
bids = orderbook.get(“bids”, [])
asks = orderbook.get(“asks”, [])
# Assume bids and asks are sorted (bids highest first, asks lowest first)
if not bids or not asks:
print(“Order book is empty or token_id invalid.”)
else:
best_bid = float(bids[0][‘price’])
best_ask = float(asks[0][‘price’])
mid = (best_bid + best_ask) / 2
# Define a range of 10% around mid
lower_limit = mid * 0.9
upper_limit = mid * 1.1
buy_liquidity = 0.0 # how much can we buy (lift asks) until price > upper_limit
for ask in asks:
price = float(ask[‘price’])
size = float(ask[‘size’])
if price <= upper_limit:
buy_liquidity += size
else:
break
sell_liquidity = 0.0 # how much can we sell (hit bids) until price < lower_limit
for bid in bids:
price = float(bid[‘price’])
size = float(bid[‘size’])
```

```
if price >= lower_limit:
sell_liquidity += size
else:
break
print(f"Mid price ~ {mid:.3f}. Within ±10% range:")
print(f" Buy-side liquidity: {buy_liquidity:.2f} shares (amount of outcome tokens available to buy before price rises 10%)")
print(f" Sell-side liquidity: {sell_liquidity:.2f} shares (amount one can sell before price drops 10%)")
```

Explanation: We retrieve the order book for a given token (likely the Yes outcome of a market). We then compute the mid price from best bid/ask. We choose an arbitrary metric: how much can be bought or sold within a 10% move of the price. We sum the size of all asks up to the price that is 10% above mid (for buy liquidity, since buying drives price up through asks), and sum all bids down to 10% below mid for sell liquidity. The result gives a sense of depth. If, say, buy_liquidity is only 100 and sell_liquidity is 500, it means there's more support on the downside than upside — one could sell 500 tokens before a 10% drop, but buying 100 tokens would push price up 10%. This indicates an imbalance in liquidity.

This is one way to interpret liquidity. One could also directly use the liquidity field from the API; Polymarket might define it differently (maybe as the sum of both sides' depth at some threshold). The liquidityNum often appears to correlate with open interest or depth*[155][156]*.

Another quick check: the openInterest field from Gamma (event.openInterest or market.openInterest) tells how many shares are in circulation. High open interest often correlates with higher liquidity since more people are involved (but not always — could be lots of held shares but not many orders in book).

Example using liquidity field directly:

```
markets = requests.get("https://gamma-api.polymarket.com/markets", params={"closed": "false", "order": "liquidity_num", "ascending": "false", "limit": 5}).json()
print("Top 5 markets by liquidity:")
for m in markets:
print(f"{m.get('question', '')[:40]:40} | Liquidity: {m.get('liquidityNum')} | Volume: {m.get('volumeNum')}")
```

Here we sorted markets by liquidity (assuming liquidity_num is sortable) and list the top 5. This quickly identifies which markets are most liquid.

4. Making Predictions (Using and Placing Data)

This phrase can be interpreted in two ways: 1. **Using market data to make predictions**: i.e., using Polymarket's prices as predictive indicators for real-world events. A developer might pull the odds and feed them into a larger model or make a decision (e.g., "the market predicts a 65% chance of X, so adjust our plans accordingly"). In this sense, "making predictions" is about interpreting the data. We've partially covered that in querying outcomes. Another example: one could create an alert if a probability crosses a threshold. Or aggregate multiple markets to predict something indirectly. 2. **Placing predictions (bets) on Polymarket via the API**: i.e., using the API to actually execute a trade, thereby "making a prediction" in the market by putting money behind it.

We will briefly illustrate both perspectives:

(a) Using API data to inform predictions: Suppose we want to predict if an event will happen and possibly hedge or take action if the probability gets high. We can set up a simple script to monitor a market's probability and trigger a message if it goes above 80%. This is straightforward with periodic calls to /price or event data.

Example: (Pseudo-code) Monitor market and log a prediction:

market_id = "abc123" # Polymarket market id or conditionId for a binary market
threshold = 0.80
while True:
data = requests.get("https://gamma-api.polymarket.com/markets", params={"id": market_id}).json()
if data:
market = data[0]
price = None
if market.get("outcomePrices"):
price = float(eval(market["outcomePrices"])[0])
elif market.get("bestBid") is not None and market.get("bestAsk") is not None:
price = (market["bestBid"] + market["bestAsk"]) / 2
if price and price > threshold:
print(f"Prediction threshold crossed! Probability = {price*100:.1f}%")

```
# e.g., send alert or take action
break
time.sleep(60) # check every minute
```

This uses Polymarket as an oracle: if the market strongly believes something (price > 0.8), we treat that as a prediction signal.

(b) Placing a trade (making a prediction on-market): Using the API to bet on an outcome. This requires authenticating and signing an order. While a full implementation is beyond scope (one needs an API key and to sign with a private key or Polymarket's key if provided), we can conceptually show how one would place an order:

Polymarket's order format (from documentation and EIP-712 structure) includes fields like: — market or conditionId — outcome (index of outcome, e.g., 0 for Yes, 1 for No in binary) — side (BUY or SELL) — price (as a fraction of 1) — size (number of shares) — possibly expiration for order, and a nonce

Polymarket requires a signature. If using an API key from the Builder program, one might do:

```
import requests, json, time, hmac, hashlib

api_key = "YOUR_API_KEY"
secret = "YOUR_SECRET"
passphrase = "YOUR_PASSPHRASE"

order = {
"market": "0x1234...condition", # the conditionId of market
"outcome": 0, # e.g., 0 for Yes
"side": "BUY",
"size": "50", # buying 50 shares
"price": "0.60", # at price $0.60 (60%)
"type": "limit",
"expiration": int(time.time()) + 3600 # expire in an hour
}
# Create the signature payload as per docs (likely HMAC of body and timestamp)
timestamp = str(int(time.time()*1000))
```

```
prehash = timestamp + 'POST' + '/order' + json.dumps(order)
signature = hmac.new(secret.encode(), prehash.encode(),
hashlib.sha256).hexdigest()

headers = {
"PM-API-KEY": api_key,
"PM-API-PASSPHRASE": passphrase,
"PM-API-TIMESTAMP": timestamp,
"PM-API-SIGN": signature,
"Content-Type": "application/json"
}
res = requests.post("https://clob.polymarket.com/order", headers=headers,
json=order)
print(res.status_code, res.text)
```

This is an *illustrative* snippet — the actual header names and signing method may differ (the above pattern is inspired by typical exchange API signing). If successful, the response could be an order confirmation with an order ID. If not, one might get an error (e.g., insufficient funds, or invalid signature if we did it wrong).

Once the order is placed, the trade (prediction) is live on Polymarket's books. If it matches, a trade occurs and the user's position updates (accessible via Data API).

Interpretation: By placing a BUY at price 0.60 for 50 shares, the user is effectively predicting the event has >60% chance. If they wanted to "make a prediction" that the event will happen, they'd buy shares; if they wanted to bet against, they'd sell or buy the No outcome. The API thus allows automated strategies — like arbitrage bots that place orders when Polymarket's price is out-of-line with another market, or market makers who continuously place both buy and sell orders around a probability.

Code Examples Summary

Throughout these scenarios, we demonstrated: — Using the Gamma API to retrieve and filter market lists. — Extracting outcome probabilities for events (using both Gamma and a bit of CLOB data). — Using the CLOB API to get order book info and derive liquidity measures. — (Conceptually) using the trading endpoint to place an order, illustrating how one would integrate the API to programmatically participate in the market.

These examples scratch the surface of what can be built. More complex applications could include: — **Automated market maker bots**: using WebSocket feeds to get real-time updates and then placing orders via the API to arbitrage or provide liquidity within certain spreads. — **Data analytics pipelines**: storing Polymarket historical data (via the API or subgraph) to analyze how effective the prediction markets are, how quickly they respond to news, etc. — **Forecast aggregation**: combining Polymarket odds with other predictors (e.g., polling data or other prediction markets) to create a consensus forecast — developers can easily pull Polymarket odds as one input.

Conclusion

The Polymarket API (latest Gamma version and associated services) offers a rich interface for blockchain developers to access and interact with decentralized prediction markets. In this paper, we have presented a thorough overview of the API's architecture, covering the distinct components: the Gamma API for market metadata and discovery, the CLOB API for live trading data and order submission, the Data API for user-specific information, and the complementary WebSocket feeds for real-time updates. Each endpoint — ranging from high-level listings like GET /markets and GET /events to granular ones like GET /price or GET /book — has been described with its purpose and output fields, effectively serving as a formal documentation of Polymarket's REST interface.

We explained how the API's data structures (markets, events, orders, etc.) relate to Polymarket's underlying smart contract framework. Notably, **market identifiers (condition IDs) bridge the API data to on-chain conditional tokens**, and fields such as bestBid, volume, or openInterest encapsulate information that is derivable from on-chain events but conveniently packaged by Polymarket's backend. The integration with UMA's Optimistic Oracle is abstracted into resolution status fields, illustrating how off-chain and on-chain components work in tandem through the API*[5]*. This design enables developers to trust but verify: they can use the API for ease of development, while references like questionID and contract addresses are available for on-chain verification if needed*[157]*.

Through use cases and code examples, we demonstrated practical applications: retrieving market odds for decision support, monitoring and analyzing liquidity, and even automating trading actions. The Polymarket API's accessibility (public read endpoints) and real-time nature (WebSockets, low-latency updates*[29]*) make it a valuable tool for a variety of projects. For instance, a research project could use the

API to study how quickly market probabilities converge to reality after news events, or a decentralized application could embed Polymarket odds as a feature (for example, a widget showing the likelihood of an outcome, powered by Polymarket data). Likewise, traders can build bots that react to market movements and either arbitrage between Polymarket and other exchanges or provide liquidity and earn rebates (Polymarket's fee structure even incentivizes such activity with maker rebates*[81]*).

From an academic perspective, Polymarket represents an ongoing experiment in crowd forecasting on the blockchain, and the API is the window into that experiment. By formally documenting the endpoints and data schema, we hope to lower the barrier for other researchers to incorporate this data into their analyses. Already, the prediction market prices have been seen as meaningful predictors in domains like politics and finance; with easy API access, one can programmatically gather evidence of the market's predictive power.

Finally, a note on **future developments**: Polymarket's API is under active development (as evidenced by frequent changelog updates*[158][159]*). Developers using this API should stay updated with Polymarket's documentation for any changes (new fields, deprecated endpoints, etc.). For example, if Polymarket introduces new market types (like scalar markets or more complex multi-outcome payouts), the API might extend to accommodate those (perhaps via new fields like lower/upperBound which already exist for scalar ranges*[83]*). Additionally, performance improvements and rate limit changes are possible; currently the API is intended for reasonable use (market makers are given higher rate limits via the builder program*[127]*). As the user base grows, Polymarket might transition more data to WebSockets or specialized endpoints to ensure scalability.

In conclusion, the Polymarket API exemplifies a well-structured approach to exposing decentralized market data. It balances the intricacies of blockchain (by providing direct links to on-chain identifiers) with the convenience of Web2 APIs (REST and WebSocket interfaces). This allows blockchain developers to build sophisticated prediction market applications without reimplementing low-level plumbing. Whether one is building a trading bot, a data analysis pipeline, or an AI that reacts to real-time probabilities, the Polymarket API provides the necessary building blocks to fetch data, interpret market signals, and execute trades. By leveraging this API, developers contribute to a broader ecosystem where

transparent, crowd-sourced predictions can inform decision-making and research across various fields.

References: The information and specifications in this paper were drawn from the official Polymarket documentation and related resources. Key sources include Polymarket's developer docs for the Gamma API*[14][160]*, CLOB API documentation*[110]*, the Polymarket Changelog for updates*[15]*, and summary guides*[161][16]*, as well as insight into on-chain interactions via Bitquery's Polymarket guide*[157]*. These sources ensure that the details presented (such as field definitions and example values) reflect the current state of the Polymarket API and its underlying system.

[1] [3] [4] [43] [49] [50] [51] [52] [91] [120] [121] [122] [123] [124] [125] [131] [132] [134] [135] [136] [137] [138] [151] [152] [161] Polymarket

https://docs.sim.ai/tools/polymarket

[2] [11] [13] [31] [144] [145] [157] Polymarket API — Get Prices, Trades & Market Data | Bitquery

https://docs.bitquery.io/docs/examples/polymarket-api/

[5] [10] Resolution — Polymarket Documentation

https://docs.polymarket.com/developers/resolution/UMA

[6] [7] [8] [9] [133] CLOB Introduction — Polymarket Documentation

https://docs.polymarket.com/developers/CLOB/introduction

[12] [32] Overview — Polymarket Documentation

https://docs.polymarket.com/developers/subgraph/overview

[14] Overview — Polymarket Documentation

https://docs.polymarket.com/developers/gamma-markets-api/overview

[15] [28] [81] [82] [116] [128] [129] [130] [146] [150] [158] [159] Polymarket Changelog — Polymarket Documentation

https://docs.polymarket.com/changelog/changelog

[16] [23] [24] [25] [26] [29] [30] [72] Data Feeds — Polymarket Documentation

https://docs.polymarket.com/developers/market-makers/data-feeds

[17] [18] [19] [20] [21] [22] [27] [44] [115] [126] [127] [143] Endpoints — Polymarket Documentation

https://docs.polymarket.com/quickstart/reference/endpoints

[33] [34] [45] [46] [47] [48] List series — Polymarket Documentation

https://docs.polymarket.com/api-reference/series/list-series

[35] [36] [153] Gamma Structure — Polymarket Documentation

https://docs.polymarket.com/developers/gamma-markets-api/gamma-structure

[37] [38] [39] [40] [53] [54] [55] [61] [62] [63] [64] [65] [71] [73] [74] [75] [76] [77] [78] [79] [80] [87] [88] [89] [90] [98] [99] [147] [148] [155] [156] [160] Get Markets — Polymarket Documentation

https://docs.polymarket.com/developers/gamma-markets-api/get-markets

[41] [42] [56] [57] [58] [59] [60] [66] [67] [68] [69] [70] [83] [84] [85] [86] [92] [93] [94] [95] [96] [97] [100] [101] [102] [103] [104] [105] [106] [107] [108] [109] [112] [154] List events — Polymarket Documentation

https://docs.polymarket.com/api-reference/events/list-events

[110] [111] [113] [114] [118] Get market price — Polymarket Documentation

https://docs.polymarket.com/api-reference/pricing/get-market-price

[117] Get midpoint price — Polymarket Documentation

https://docs.polymarket.com/api-reference/pricing/get-midpoint-price

[119] Historical Timeseries Data — Polymarket Documentation

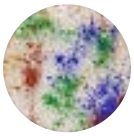
https://docs.polymarket.com/developers/CLOB/timeseries

[139] *[140]* *[141]* *[142]* Get Market Holders (Data-API) — Polymarket Documentation

https://docs.polymarket.com/developers/misc-endpoints/data-api-holders

[149] py-clob-client — PyPI

https://pypi.org/project/py-clob-client/



Follow

Written by Jung-Hua Liu

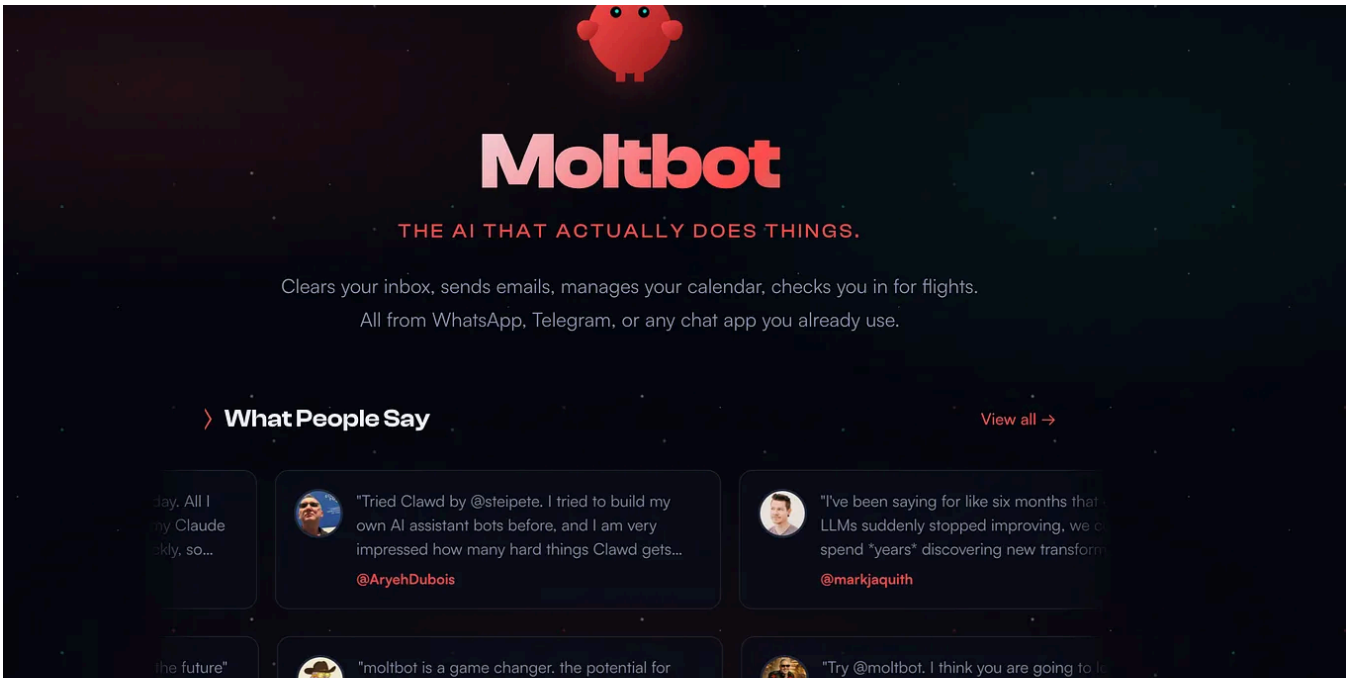
310 followers · 11 following

No responses yet

Write a response

What are your thoughts?

More from Jung-Hua Liu



Jung-Hua Liu

Clawdbot (Moltybot): A Self-Hosted Personal AI Assistant and Its Viral Rise

Introduction

Jan 27 55

Type	Examples	Tools/Libraries
rical Prices /JSON)	Yahoo Finance (BTC-USD), CCXT Binance OHLCV	yfinance, ccxt, pandas, databases (SQL)
hain Metrics	Ethereum transaction counts, gas usage, token transfers	Blockchain APIs (Etherscan), web3.
epapers & Docs	Bitcoin/Ethereum whitepaper (PDF), protocol specifications	requests + PDF parsers (PyMuPDF pdfminer), LlamaIndex DocumentLoaders
s Articles HTML/RSS)	CoinDesk, CoinTelegraph, Finance RSS feeds	feedparser, newspaper3k, requests+BeautifulSoup
time Market s	Binance/Coinbase live trades, price tick updates	WebSockets (e.g. websocket-clie ccxt streaming

Jung-Hua Liu

Integrating LLM APIs, Document Extraction, and RAG Pipelines for Cryptocurrency Data Analysis

Abstract. We present a comprehensive framework for analyzing cryptocurrency data using large language models (LLMs), agentic frameworks...

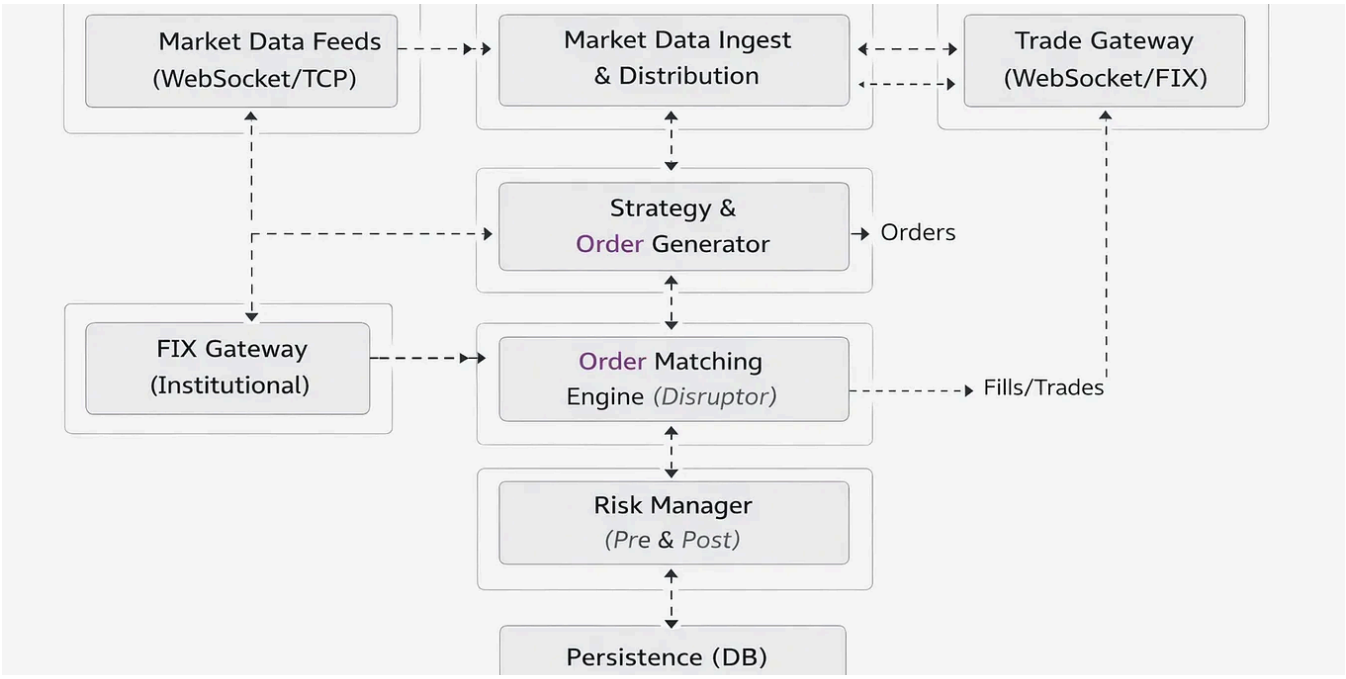
Jan 13 3

Jung-Hua Liu

Top 10 AI-Powered Crypto Trading Repositories on GitHub

Below is a ranked list of the top 10 GitHub repositories focused on cryptocurrency trading strategies that incorporate AI techniques (e.g...

Jun 22, 2025 20 2



Jung-Hua Liu

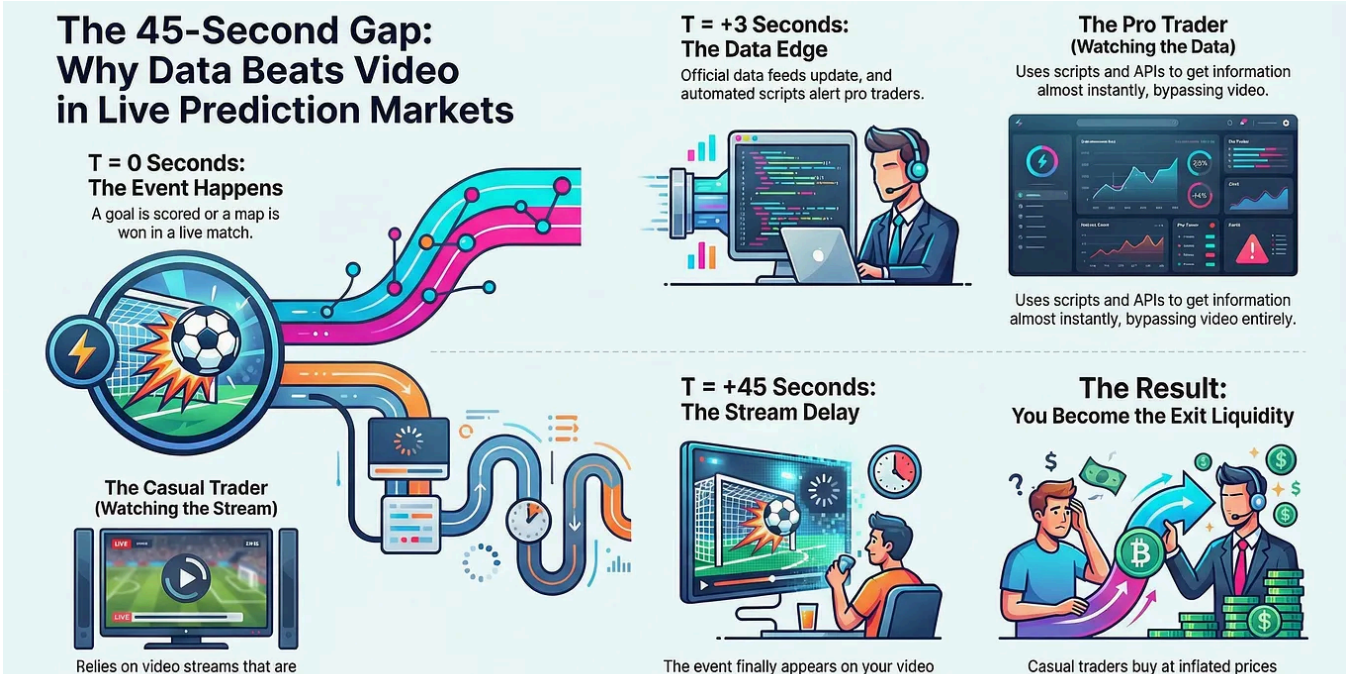
Design and Implementation of a Low-Latency High-Frequency Trading System for Cryptocurrency Markets

Abstract

Jan 24 47

See all from Jung-Hua Liu

Recommended from Medium



Ezekiel Njuguna

How Smart Traders Beat You on Polymarket Live Markets

Created by ezzekiel the writer

Jan 7 111 1

In InsiderFinance Wire by Sips & Scale

Will This Be Your Next Polymarket Bot?

How Polymarket Bots Really Make Their Money. A realistic tour of tiny edges, fast code, and real risk.

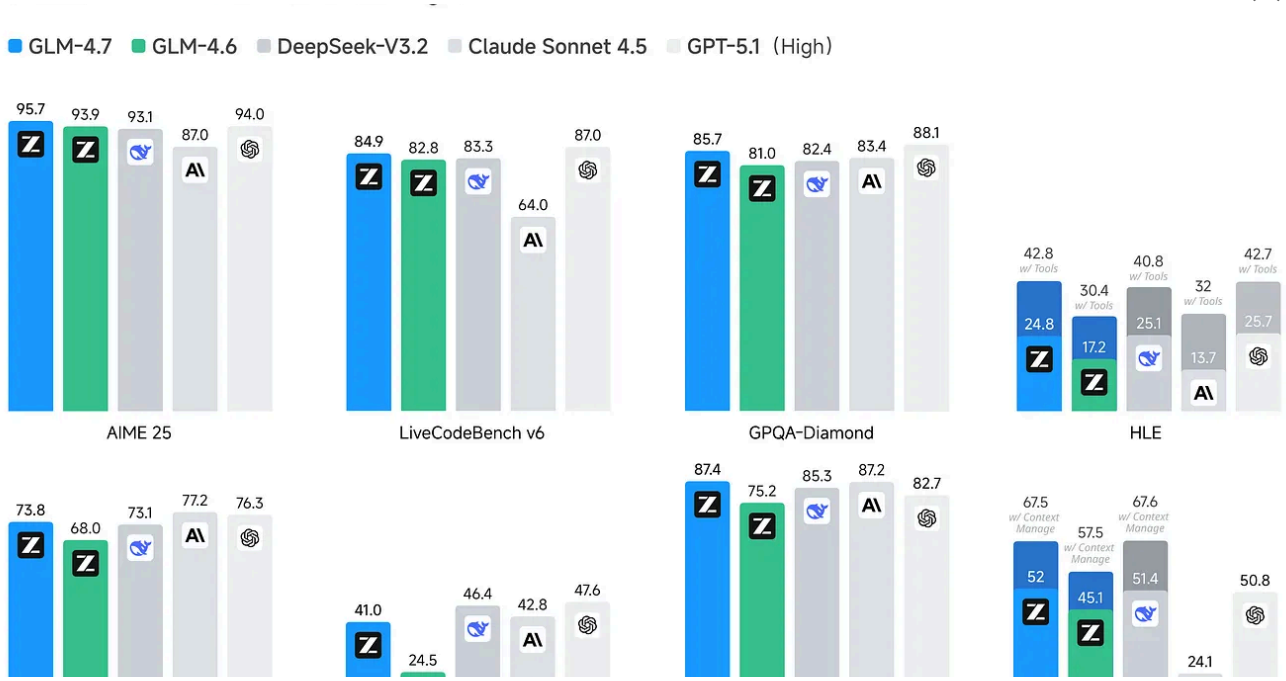
Jan 22 73

Joe Njenga

I Tested Kimi K2.5 with Claude Code (1-Trillion Parameters, 8x Cheaper Than Opus)

Moonshot AI never stops surprising us— Kimi K2.5 is out, so I paired it with Claude Code, but using Ollama.

Jan 28 522 12



Agent Native

Local LLMs That Can Replace Claude Code

Small team of engineers can easily burn >$2K/mo on Anthropic's Claude Code (Sonnet/Opus 4.5). As budgets are tight, you might be wondering...

Jan 20 736 23

Dexoryn || Web3 + AI builder

7 Polymarket Arbitrage Strategies Every Trader Should Know

By Dexoryn — Trading Bot & Casino Game Developer GitHub: https://github.com/dexorynlabs/polymarket-trading-bot-ts

Jan 22 2

In CoinsBench by Michal Stefanow

Inside the Mind of a Polymarket BOT

If you have ever opened a Bitcoin 15-minute market on Polymarket and wondered why one trader always seems to walk away with a win, this is...

Dec 4, 2025 111 94

See more recommendations