ChatGPT

API Endpoint Framework: Velo (Pro), Dune Analytics, CoinGecko Pro, DefiLlama Pro

Velo Pro API (Aggregated Crypto Data)

Velo is a paid API service ($199/mo) providing cross-exchange crypto market data (futures, options, spot) in high resolution [1] [2]. With a Pro API key, you can query historical and real-time data across multiple exchanges and instruments. Key endpoint categories include:

- **Helper Endpoints:**
- `/status` – Check API key status (and quota usage if applicable).
- `/futures` – List all supported futures contracts (with exchange and symbol identifiers) [3] [3].
- `/options` – List all supported options (e.g. tickers or symbols) [3].
- `/spot` – List all supported spot trading pairs (exchange and pair info) [3].
  (The above are public endpoints to discover available products/histories; no auth needed)

- **Data Endpoints:**

- `/rows` – Core data query endpoint. Returns time-series rows of market data (in CSV format) for a given `type` (one of `'futures'`, `'options'`, `'spot'`) [4]. You can specify parameters:
  - `exchanges` (one or more exchange names, comma-separated),
  - `products` (specific instruments or trading pairs), or `coins` (asset symbols) – *if both are given,* `products` *takes priority* [5],
  - `columns` – which data fields to retrieve (price, volume, open interest, etc.),
  - `begin` and `end` – UNIX timestamps in milliseconds to define the time range,
  - `resolution` – interval resolution (e.g. `'1m'` for 1-minute) [6].

  Output: CSV text with a timestamp column and the requested data columns. For example, a query for spot prices might return:

```
time,close_price
1704057600000, 29350.12
1704057660000, 29347.09
... (up to 22,500 rows per request) ...
```

  Note: Each request is capped at 22,500 data points; if you need more history, implement batching/pagination by splitting the time range [7]. The available `columns` for each market type include typical metrics like `open_price, high_price, low_price, close_price, coin_volume, dollar_volume, funding_rate`, etc. (full lists are documented [8] [9]).

- `/terms` – Options term structure (implied volatility skew) data. Accepts `coins` parameter (e.g. `coins=BTC,ETH`) and returns the latest options term structure for those assets in CSV format [10] (each term's IV and skew values).

- `/caps` – Market capitalization data. Accepts `coins` list and returns latest market cap for each specified coin (CSV) [11]. This is useful for quickly fetching current market cap and other aggregated stats.

- **News Endpoint:**

- `/news` – Returns recent crypto news and on-chain event headlines (in JSON). You can include an optional `begin=<timestamp>` to fetch news after a certain time [12] [13]. Each news object contains fields like `id`, `time` (publish timestamp), `headline`, `source`, `priority` (importance), related `coins`, `summary` (often in Markdown), and `link` [14]. Example JSON news item:

```
{
  "id": 55,
  "time": 1704085200000,
  "headline": "Hello world",
  "source": "Velo",
  "coins": ["BTC"],
  "summary": "# Hello world",
  "link": "https://velodata.app"
}
```

The News API also supports WebSocket streaming for live updates [15] (useful if building real-time dashboards of news sentiment).

Authentication: Velo uses HTTP Basic Auth. Provide your API key via basic authentication with username "`api`" and password as your API key [16]. For example, with cURL:

```
curl -u "api:<YOUR_API_KEY>" "https://api.velo.xyz/api/v1/caps?coins=BTC,ETH"
```

In code, set the Authorization header accordingly. (No separate API key parameter in the URL – the key **must** be passed via basic auth or Velo's official SDK clients.)

Example – Fetching Historical Price Data: To retrieve, say, the last 24 hours of 1-minute BTC spot prices from a specific exchange, you could call:

```
GET https://api.velo.xyz/api/v1/rows?type=spot&exchanges=coinbase&products=BTC-USD&
    columns=close_price&begin=1703980800000&end=1704067200000&resolution=1m
    (with Basic Auth header "api:YOUR_API_KEY")
```

This returns a CSV where each row has a timestamp and BTC-USD closing price each minute within the range.

Rate Limits & Usage: Velo's documentation does not explicitly publish a rate limit, but given the Pro tier, it's designed for high-volume use. The `/status` endpoint can be used to check your key's status (and possibly remaining quota) [17]. Expect that heavy usage is allowed (as the subscription is paid), but for extremely large data pulls you should implement request batching due to the 22.5k-row limit per call [7]. If you need full history (beyond 3 months), ensure you have the yearly subscription (which unlocks the entire historical dataset) [1].

Data Schema & Field Explanations: Velo returns data in a structured table format (CSV or as Python/Node structures via their SDK). Common fields: - **Time** – UNIX epoch in milliseconds (all data points time-keyed in UTC). - **Price fields:** `open_price, high_price, low_price, close_price` – OHLC prices for the interval [18]. - **Volume fields:** `coin_volume` (volume in base asset units), `dollar_volume` (volume in USD) [19]. - **Trade counts:** `buy_trades, sell_trades, total_trades` – number of trades in that interval [19]. - **Open Interest (futures/options):** e.g. `coin_open_interest_close`, `dollar_open_interest_close` – open interest at close of interval (in coins and USD) [20]. - **Funding rates (futures):** `funding_rate` (most recent interval funding, normalized to 8h) and `funding_rate_avg` [21]. - **Options-specific:** implied volatilities (`iv_1w, iv_1m, ...` for 1-week, 1-month, etc), skew, option Greeks and aggregated volume/notional values [22] [23]. - **Market cap (from `/caps`):** likely returns columns like `coin, time, market_cap` for each requested coin (with `time` being the timestamp of the latest update, and `market_cap` in USD).

Note: Velo's API is flexible but raw – it returns data as CSV text for efficiency. In a Replit environment, you might use Python's `requests` to GET data and `csv` or pandas to parse it, or use Velo's official Python client (`velodata` library) for convenience [24]. The data can then be fed into your dashboard charts.

Dune Analytics API (Pro Level)

Dune Analytics offers a powerful API to query blockchain data using SQL or pre-built endpoints. With a Dune API key (available to Dune Pro/Business users), you can programmatically run queries and retrieve results for on-chain metrics, token analytics, smart contract data, and more [25]. The API is divided into **low-level SQL endpoints** (fully customizable queries) and **high-level preset endpoints** (pre-built datasets requiring no SQL). Key components:

- **Custom Query & SQL Endpoints:** These allow you to execute arbitrary Dune **SQL queries** and get results:
- **Query Execution:** `POST /api/v1/query/{query_id}/execute` – Runs a saved Dune query by ID [26]. You supply the query ID (which corresponds to a query on dune.com) and Dune returns an `execution_id`.
- **Get Results:** After execution, retrieve results via `GET /api/v1/execution/{execution_id}/results` (with optional `?download` or format flags). Results can be fetched in JSON (default) or CSV. Alternatively, Dune provides a convenience endpoint `GET /api/v1/query/{query_id}/results` for recent cached results if available.

- **Query Management:** There are also endpoints to list or create queries (`/api/v1/queries`), manage query definitions, and even set up **webhooks** for query result updates [27] – useful for scheduling updates to your dashboard.
- **Pagination:** Dune queries that return large result sets will be paginated. The results JSON includes `next_offset` and a `next_uri` if not all rows are returned in one go [28]. You can append `?limit=<N>&offset=<M>` to the results endpoint to page through. By default, the API might return the first 100 rows (for example) – you can request a larger `limit` up to some maximum (commonly 1000) per call. The presence of `next_uri` in the response indicates more data is available [29].

Example: Running a custom query:

```
# Start execution of a query (POST request)
curl -X POST "https://api.dune.com/api/v1/query/123456/execute" \
     -H "X-DUNE-API-KEY: <YOUR_API_KEY>"
```

This returns a JSON with an `execution_id` and an initial status (e.g. QUEUED or RUNNING). You would then poll:

```
curl -X GET "https://api.dune.com/api/v1/execution/<EXECUTION_ID>/results" \
     -H "X-DUNE-API-KEY: <YOUR_API_KEY>"
```

If the query has completed, you'll get a JSON `state: QUERY_STATE_COMPLETED` and a `result` object. If not, you may get a `state` like `QUERY_STATE_RUNNING` and need to retry after a short delay.

- **Preset Data Endpoints:** Dune provides **ready-made endpoints** for popular data without writing SQL. These are essentially **curated queries** maintained by Dune, covering topics such as:
- **Decentralized Exchange stats (DEX):** e.g. `GET /api/v1/dex/pairs/{chain}` – Returns metadata and stats for all token pairs on a given chain (24h/7d/30d volumes, liquidity, etc) [30] [31]. There are also endpoints like `/api/v1/overview/dexs` and `/api/v1/overview/dexs/{chain}` for aggregated DEX volumes [32] [33], and `/api/v1/summary/dexs/{protocol}` for a specific DEX's volume history [34].
- **NFT & Markets:** `GET /api/v1/markets` – Market share of NFT marketplaces vs DEXs (for instance), or endpoints to track NFT trading stats (if provided). *(Dune's "Markets" preset covers cross-market metrics like NFT vs DEX volume share [35].)*
- **Trending Contracts:** `GET /api/v1/contracts` – Lists trending smart contracts on EVM chains (based on activity in last 30 days) [35].
- **EigenLayer, Farcaster, Projects:** Various endpoints for specialized data (EigenLayer restaking metrics, Farcaster social data, specific DeFi project dashboards, etc) [36].
- **Projects/Protocols:** Under "Projects" presets, you may find endpoints for specific protocols (Dune sometimes publishes API endpoints for popular protocols' stats).

Each preset endpoint has its own path. For example, the **DEX pair stats** endpoint above returns a JSON like:

```
{
  "execution_id": "01H...Q1",
  "state": "QUERY_STATE_COMPLETED",
  "result": {
    "metadata": {
      "column_names": ["token_pair","one_day_volume","usd_liquidity", ...],
      "row_count": 10,
      ...
    },
    "rows": [
      {
        "token_pair": "USDC-WETH",
        "one_day_volume": 12345678.9,
        "usd_liquidity": 50123456.7,
        "...": "..."
      },
      { ... more rows ... }
    ]
  },
  "next_uri": "https://api.dune.com/api/v1/execution/01H...Q1/results?
offset=10&limit=10"
}
```

(In many cases, the `rows` might be an array of objects as above, or an array of arrays with separate `column_names` – Dune's API recently moved toward returning an array of objects for preset endpoints, as shown, making it easier to interpret.) The `metadata` section describes the result set: column names, types, number of rows, etc [37]. If there are more than 10 rows in this example, `next_uri` provides the link to fetch the next page.

- **Data Management Endpoints:** Dune also allows uploading custom data tables and managing them (for advanced use) [38]. This is likely outside the scope of a read-only dashboard, but if needed, Pro API keys can create tables via `/api/v1/table/{name}` etc. You can typically ignore these for visualization purposes unless you plan to push your own data to Dune.

Authentication: All Dune API calls require an API key in the header or query. **Use the HTTP header method**: include `X-DUNE-API-KEY: <YOUR_API_KEY>` in each request header [39] [40]. (Alternatively, you can append `?x-dune-api-key=<YOUR_API_KEY>` to the URL, but this is less secure and not recommended for client-side requests [41] [42].) API keys are generated from your Dune account settings (each key is scoped to either your user or a specific team context) [43]. Keep your key secret.

Rate Limits: The Dune API is **rate-limited by your plan**. For example, Free plans might have ~15 low-priority and 40 high-priority requests/minute [44], while paid **Premium** plans allow up to ~1000 requests/minute [45]. Pro users typically get a significantly higher rate limit and monthly query executions. Check Dune's docs or FAQ for your specific plan's limits [45]. Dune classifies queries as *heavy* or *light*; simple preset calls may count differently than full query executions. Always handle HTTP 429 responses (rate limit exceeded) by backing off or reducing frequency.

Pagination & Query Size Limits: Dune enforces an 8 GB maximum result size for any query [46]. For very large query results, consider adding `LIMIT` clauses in SQL or filtering. The API's pagination parameters (`limit` and `offset`) allow you to retrieve large result sets in chunks [28]. For example, if a query returns 10k rows, you might fetch `?limit=1000` and loop through `offset=0,1000,2000,...` until all rows are fetched.

Data Schema: Since Dune is flexible, the schema depends on the query. In general: - Preset endpoints return structured JSON with `result.metadata` (column names, data types, row count, etc) and `result.rows` (array of results). Each row might be an object with key-value pairs for each column (as shown above), or an array aligned with `column_names`. Check the docs for each endpoint's exact format. - **Timestamp formats:** Dune often returns timestamps as ISO strings (e.g. `"2024-12-20T11:04:18.724Z"`) or Unix time depending on the query. - **Numeric types:** Large integers (like token amounts) may be returned as strings to avoid precision issues (common in JSON APIs). - **Example fields:** A result for a DEX volume query might include fields like `volume_usd`, `volume_eth`, `trade_count`, etc. A query for active users might return `user_address` and `last_active_date`. Always refer to the column names in `metadata.column_names` for what each index in `rows` represents [37].

Using Dune's API in Replit is suitable if you want to display custom on-chain analytics or data not readily available via other APIs. For instance, you could create a Dune query for "daily active users of Protocol X" or "smart contract call counts for Contract Y" and then use the Dune API to fetch those results periodically for your dashboard. The **Dune client libraries** (available in Python, JavaScript, etc.) can simplify some tasks [47] [48], but simple `fetch`/`requests` calls with the API key header work just as well.

CoinGecko Pro API (Crypto Market Data)

CoinGecko provides a comprehensive REST API for cryptocurrency market data. The Pro API (paid plans) offers the same endpoints as the free API plus additional data (e.g. on-chain DEX data) and higher rate limits/quotas. All endpoints return **JSON** data. The base URL for Pro is `https://pro-api.coingecko.com/api/v3/` [49]. Below is a categorized map of key endpoints:

- **Ping & Key Info:**
- `GET /ping` – Simple service ping. Returns `{"gecko_says": "(...)"}` to confirm the API is reachable [50].

- `GET /key` – Account API usage info. Returns your usage and rate-limit status [50]. Use this to monitor monthly credits, remaining calls, etc.

- **Simple Data:**

- `GET /simple/price` – Get current price of one or more coins in one or more currencies [51]. You supply coin IDs and target currencies (e.g. ids=bitcoin,ethereum & vs_currencies=usd,eur). Returns a JSON mapping coin IDs to prices, e.g. `{"bitcoin":{"usd":29000, "eur":27000}, "ethereum":{"usd":1800,...}}`.
- `GET /simple/token_price/{id}` – Price for tokens by *platform contract address*. For example, `/simple/token_price/ethereum?contract_addresses=0xdAC...&vs_currencies=usd` gives price for a specific ERC-20 token [52].

- `GET /simple/supported_vs_currencies` – List of currency codes supported (for conversion rates) [53] .

- **Coins (Market Data & Metadata):** *(These are among the most used endpoints)*

- `GET /coins/list` – List all coins with their `id`, `symbol`, and name [54] . The `id` here is CoinGecko's internal ID (used in other endpoints).

- `GET /coins/markets` – Market data for a list of coins [55] . You provide parameters like `vs_currency` (e.g. usd), optional `ids` (list of coin IDs to limit the scope), `order` (e.g. market_cap_desc), `per_page` and `page` for pagination, and `price_change_percentage` to include 1h/24h/7d change percentages. The response is an array of objects, each with fields:

    - `id`, `symbol`, `name` – identifiers,
    - `image` – URL of coin's logo,
    - `current_price`, `market_cap`, `total_volume`, `circulating_supply`, etc.,
    - `high_24h`, `low_24h`, `price_change_24h`, `price_change_percentage_24h` (and similarly for 7d etc if requested) – price movement stats,
    - `market_cap_rank` – rank by market cap,
    - other fields like `ath` (all-time-high) and `atl` values with dates, etc.
    Example (truncated) for one coin:

    ```
    {
      "id": "bitcoin",
      "symbol": "btc",
      "name": "Bitcoin",
      "image": "https://assets.coingecko.com/coins/images/1/large/
    bitcoin.png?...",
      "current_price": 29012.45,
      "market_cap": 564839000000,
      "market_cap_rank": 1,
      "total_volume": 34215000000,
      "high_24h": 29350.12,
      "low_24h": 28500.34,
      "price_change_24h": 512.11,
      "price_change_percentage_24h": 1.80,
      "price_change_percentage_7d_in_currency": 3.5,
      "circulating_supply": 19400000,
      "total_supply": 21000000,
      "ath": 69045,
      "ath_date": "2021-11-10T14:24:11.849Z",
      "...": "..."
    }
    ```

This endpoint is paginated (`per_page` up to 250; use `page=2` etc. for more coins). Pro tip: if you need **all** coins' data, iterate through pages or use the `/coins/list` then query in batches.

- `GET /coins/{id}` – Full coin data by ID [56]. Returns a detailed JSON for the coin: market data, community data, developer data, descriptions, links, tickers, etc. This is quite extensive (including fields like `block_time`, `hashing_algorithm`, categories, and even localization of the description in multiple languages). Use this for coin metadata and extended stats.

- `GET /coins/{id}/tickers` – All trading pairs for the coin on various exchanges (with prices, volume, etc) [57].
- `GET /coins/{id}/history?date=DD-MM-YYYY` – Historical data for a specific date [58] (a snapshot of price, market cap on that date).
- `GET /coins/{id}/market_chart?vs_currency=X&days=N` – Historical daily chart data (price, market cap, volume) for the past N days [59]. `days` can be a number or the string `max` for full history. The response has `prices`, `market_caps`, `total_volumes` as arrays of `[timestamp, value]` pairs (timestamps in milliseconds) – perfect for plotting time-series.
- `GET /coins/{id}/market_chart/range?vs_currency=X&from=UNIX_TIME&to=UNIX_TIME` – Same as above but for a custom date range [60]. Use this if you need specific time windows.
- `GET /coins/{id}/ohlc?vs_currency=X&days=N` – Candlestick OHLC data for the past N days [61]. Returns an array of `[time, open, high, low, close]` for each day (or each hour if small range).
- `GET /coins/{id}/ohlc/range?...` – OHLC for a custom range (if supported) [62].

- **Supply endpoints:** For certain coins, Pro API provides supply over time:

    - `/coins/{id}/circulating_supply_chart?days=N` (and `/.../range`) – historical circulating supply [63].
    - `/coins/{id}/total_supply_chart?days=N` (and range) – historical total supply [64].

- **Coin by Contract:** If you have a token contract address on a known chain, use `/coins/{platform_id}/contract/{contract_address}` to get that token's CoinGecko data [65] (including its CoinGecko ID if it exists). Historical chart endpoints exist for contracts too [66].

- **Categories & Listings:**

    - `GET /coins/categories` – All coin categories (DeFi, NFTs, layer1, etc) with their market caps and volumes [67].
    - `GET /coins/categories/list` – Just the list of category names and IDs [67].
    - `GET /coins/top_gainers_losers?duration=daily` – *(Pro-only)* Top 30 gainers and losers for a given duration (e.g. daily, weekly) [68].
    - `GET /coins/list/new` – Latest listed coins (last ~200 new coins) [68].

- **Exchanges & Derivatives:**

- `GET /exchanges` – List of exchanges with basic info (id, name, year established, etc) and latest volume data [69].

- `GET /exchanges/{id}` – Details about a specific exchange (including centralized vs decentralized, trust score, trade volume breakdown) [70] .
- `GET /exchanges/{id}/tickers` – All trading pairs on that exchange [71] .
- `GET /exchanges/{id}/volume_chart?days=N` – Daily volume history (in BTC or USD) for that exchange [72] (`/volume_chart/range` for a specific date range [72]).
- `GET /exchanges/list` – ID and name of all exchanges (for mapping).

- Similar endpoints exist for **Derivatives**:

    - `GET /derivatives` – List current futures/options tickers and prices [73] .
    - `GET /derivatives/exchanges` – List of derivatives exchanges and aggregated data (open interest, 24h volume, etc) [73] .
    - `GET /derivatives/exchanges/{id}` – Details and instruments on a specific derivatives exchange [74] .
    - `GET /derivatives/exchanges/list` – All derivatives exchange IDs and names.

- **NFTs (Beta):** *(Pro plans only)*
  CoinGecko provides NFT collection data:

- `GET /nfts/list` – All supported NFT collections (with id, name, asset_platform_id, contract address) [75] .
- `GET /nfts/{id}` – Details for an NFT collection (floor price, market cap, volume, number of items, etc) [76] .
- `GET /nfts/{id}/market_chart?days=N` – Historical floor price and volume chart for that collection [77] .

- `GET /nfts/{id}/tickers` – Latest floor prices on each marketplace (e.g. OpenSea) [78] .
  (You can also query by contract address via `/nfts/{platform_id}/contract/{contract_address}` *similarly to coins.)*

- **Global & Misc:**

- `GET /global` – Global crypto data (total market cap, volume, market dominance percentages, number of active coins and markets) [79] .
- `GET /global/decentralized_finance_defi` – Global DeFi metrics (DeFi market cap, DeFi volume, etc) [80] .
- `GET /global/market_cap_chart?days=N` – Global total market cap chart over last N days [81] [82] .
- `GET /exchange_rates` – Current fiat and crypto exchange rates (value of 1 BTC in many currencies) [83] .
- `GET /search` – Search for coins, categories, and exchanges by name [84] .
- `GET /search/trending` – Top trending search results (returns currently hot coin IDs on CoinGecko) [85] .

- `GET /companies/public_treasury/{coin_id}` – Companies holdings for BTC or ETH [86] (e.g., shows how much BTC public companies hold, if querying BTC).

- **On-Chain DEX Data (GeckoTerminal integration, Beta):** These endpoints (prefixed with `/onchain`) give DeFi DEX and liquidity pool data, which is a newer addition to CoinGecko Pro:

- **Networks & Dexes:** `GET /onchain/networks` – List supported blockchain networks (IDs) in GeckoTerminal [87]. `GET /onchain/networks/{network}/dexes` – List DEXes on a given network [88].
- **Pools:** `GET /onchain/networks/{network}/pools` – Top pools on that chain (by liquidity) [89]. `GET /onchain/networks/{network}/new_pools` – Newly added pools on that chain [90]. `GET /onchain/networks/{network}/dexes/{dex}/pools` – top pools on a specific DEX [91]. Specific pool data: `GET /onchain/networks/{network}/pools/{pool_address}` – detailed info for a pool (reserves, volume, fees, etc) [92]. You can also batch: `GET /onchain/networks/{network}/pools/multi/{pool1},{pool2},...` to fetch multiple pools at once [92].
- **Tokens:** `GET /onchain/networks/{network}/tokens/{token_address}` – token data on that chain (price, liquidity, volume across pools) [93]. Batch: `/tokens/multi/{addr1},{addr2},...` [93]. `GET /onchain/networks/{network}/tokens/{token_address}/info` – token metadata (name, symbol, links, etc) [94].
- **Prices & OHLCV:** `GET /onchain/simple/networks/{network}/token_price/{token_address}` – current price of a token by address [95]. `GET /onchain/networks/{network}/pools/{pool_address}/ohlcv/{period}` – historical OHLCV for a pool (period could be daily etc) [96].
- **Trades:** `GET /onchain/networks/{network}/pools/{pool_address}/trades` – recent trades in that pool (past 24h or specified range) [97].
- **Trending & Search:** `GET /onchain/networks/trending_pools` (across all chains) and `/onchain/networks/{network}/trending_pools` – trending liquidity pools [88]. `GET /onchain/search/pools?query=XYZ` – search pools by name/token. `GET /onchain/pools/trending_search` – top searched pool queries [98].
- **Categories:** `GET /onchain/categories` – All DeFi categories/tags available in GeckoTerminal [99]. `GET /onchain/categories/{id}/pools` – all pools under a specific category (e.g., "Stablecoin Pools") [99].
- These on-chain endpoints let you build dashboards for DEX liquidity, yield farming, etc., similar to what DefiLlama offers, but via CoinGecko's data (powered by GeckoTerminal). **Note:** Include the `/onchain` path and your API key for these calls [100] [101].

Authentication & API Keys: For **Pro API**, you must include your API key with each request. CoinGecko supports two methods [42]: 1. **HTTP header** – **Recommended** for security. Send `x-cg-pro-api-key: YOUR_API_KEY` in the request header [102]. 2. **Query parameter** – Append `?x_cg_pro_api_key=YOUR_API_KEY` to the URL [103]. (This is easier for quick tests, but be careful not to expose the key in client-side code or URLs.)

Ensure you use the **Pro base URL** (`pro-api.coingecko.com/api/v3`). If you call the regular api.coingecko.com with a key, it won't be recognized. It's good practice to keep your key on the backend (e.g., in Replit's secrets) and proxy requests if building a public web dashboard [104].

Rate Limits & Pagination: Paid plans come with higher rate limits and monthly credit allocations. For instance, CoinGecko's **Pro** plan might allow on the order of hundreds of calls per minute and hundreds of thousands of calls per month (exact numbers depend on the plan; CoinGecko's Developer Dashboard shows your usage and limits). Every request (status 200) counts as one credit towards your monthly allotment [105].

Even a 4xx/5xx error counts against the **per-minute** rate limit (though not against monthly credits) [106] . You can check your usage with the `/key` endpoint or on the dashboard.

Most GET endpoints do not enforce pagination via tokens but rather via `page` and `per_page` parameters. For example, to get all coins on `/coins/markets`, loop page=1,2,3… until an empty array is returned. Some endpoints (like `coins/list`) return all data in one call (which can be large). The Pro API documentation notes that certain endpoints might have data or parameters exclusive to higher plans [107] (e.g., more granularity or extended history might be enterprise-only). In the docs, endpoints marked with a crown are for paid users [108] , and those with diamond for enterprise.

Data Schema & Field Notes: CoinGecko's fields are generally self-explanatory: - Prices are usually in your requested vs_currency (e.g., `usd`). - Percent changes like `price_change_percentage_24h` are given in percent (%). For longer periods (7d, 30d, 1y) you must request them and they'll appear as e.g. `price_change_percentage_7d_in_currency` [109] . - Timestamps are often in ISO8601 strings (e.g., `"last_updated": "2025-07-18T09:30:45.123Z"`). Chart data uses Unix timestamps (ms since epoch). - If a field is not applicable, it might be null. E.g., `total_supply` might be null for coins with uncapped supply. - On-chain DEX endpoints often return arrays of objects for lists of pools or tokens, including fields like `tvl_usd`, `volume_24h`, `apy` (if relevant), etc., similar to DefiLlama's data.

For visualization, common patterns include: - Use `/coins/markets` for a quick list of top coins and their daily percent changes, - Use `/coins/{id}/market_chart` for plotting price over time, - Use `/global` for a high-level market cap/volume chart, - Use `/onchain/.../pools` and `/onchain/.../tokens` to analyze DeFi liquidity.

Example – Multi-Coin Price Data: To get historical prices for multiple tokens to plot on a multi-line chart, you could call CoinGecko's `/market_chart` for each token. For instance:

```python
# Pseudo-code for Python
coins = ["bitcoin", "ethereum", "solana"]
for coin in coins:
    url = f"https://pro-api.coingecko.com/api/v3/coins/{coin}/market_chart"
    params = {"vs_currency":"usd", "days": 30}
    headers = {"x-cg-pro-api-key": API_KEY}
    data = requests.get(url, headers=headers, params=params).json()
    prices = data["prices"]  # list of [timestamp, price]
    # Process or normalize prices for chart...
```

This gives you daily (default) prices for 30 days for each coin. You could also use `market_chart/range` for exact alignment by timestamps. (If you have many coins, beware of rate limits; you might need to space out or parallelize with caution.)

CoinGecko's **Pro API** is reliable for market data and is likely where you get **price feeds, market caps, volumes, and percent changes** for your dashboard.

DefiLlama Pro API (DeFi Metrics & Aggregated Data)

DefiLlama is known for aggregating DeFi data – TVL (Total Value Locked), yields, revenue, etc. The **Pro API** (paid ~$300/mo [110]) offers **all the data categories** that DefiLlama tracks, with higher rate limits (up to 1000 req/min, 1M calls/month) [111]. This API is ideal for deep DeFi analytics and on-chain metrics. Major endpoint categories:

- **TVL & Protocols:**
- `GET /api/protocols` – List all protocols with their current TVL and basic info [112] [113]. Each protocol entry includes fields like `name`, `slug (id)`, `tvl`, and possibly breakdowns or tags.
- `GET /api/tvl/{protocol}` – **Simplified TVL**: returns the latest total value locked for a given protocol as a single number [114] [115]. Useful for quick checks or sparkline charts.
- `GET /api/protocol/{protocol}` – Detailed protocol data [116] [117]. This typically returns historical TVL time series for the protocol, broken down by chain and/or by asset. For example, you might get a structure with `tvl` over time, and maybe nested data for each chain (if multi-chain). Also includes other metadata (like the protocol's category, logo URL, etc).
- `GET /api/categories` – List of all protocol categories (e.g., Lending, DEXes, Yield Aggregators) with aggregate TVL [118].
- `GET /api/forks` – List of protocol "forks" (e.g., showing relationships like Sushi is a fork of Uniswap) [119].

- `GET /api/oracles` – List of oracles used by protocols [120].

- **Chains & Global DeFi:**

- `GET /api/v2/chains` – TVL of all chains (blockchains) currently tracked [121]. This returns an array of chain objects with fields like `name`, `tvl`, etc., giving you the total DeFi TVL per chain.
- `GET /api/v2/historicalChainTvl` – Overall historical DeFi TVL (across all chains) over time [122]. This is essentially the growth of DeFi.
- `GET /api/v2/historicalChainTvl/{chain}` – Historical TVL for a specific chain [123].
- `GET /api/chainAssets` – Assets held on each chain [124]. This might return data on how much of each major asset is on a chain (e.g., how much USDC on Ethereum vs Polygon, etc).
- `GET /api/activeUsers` – Active user counts on DefiLlama's tracked protocols and chains [125] [126]. Likely returns a summary of user activity (possibly the number of distinct addresses interacting, etc).

- `GET /api/userData/{type}/{protocolId}` – Specific user stats for a protocol [127]. For example, `type=activeUsers` might give daily active users for that protocol, if available.

- **Stablecoins:** (DefiLlama has an extensive stablecoin dashboard, and the API reflects that)

- `GET /stablecoins/stablecoins` – List all stablecoins with current circulating amounts [128]. Fields include each stablecoin's symbol, supply, perhaps peg info, etc. `include_prices=true` can add price info (to see slight depegs) [129].
- `GET /stablecoins/stablecoindominance/{chain}` – Stablecoin dominance on a chain [130] – returns what % of chain TVL is stablecoins, and which stablecoin is largest, etc.

- `GET /stablecoins/stablecoinchains` – Current total stablecoin TVL per chain (how much value in stablecoins on each chain) [131] .
- `GET /stablecoins/stablecoincharts/all` – Historical market cap of all stablecoins over time (global chart) [132] .
- `GET /stablecoins/stablecoincharts/{chain}` – Historical stablecoin market cap on a specific chain [133] .
- `GET /stablecoins/stablecoin/{id}` – Historical data for a specific stablecoin (market cap over time, and its breakdown across chains) [134] .

- **Yield Farming & Pools:**

- `GET /yields/pools` – Latest data for all yield farms/pools DefiLlama tracks (this is a large list) [135] . Each pool entry has details like platform, chain, APY, TVL, etc. *Pro API might include additional fields like projected yields.*
- `GET /yields/chart/{pool}` – Historical TVL and APY for a specific yield pool (identified by an ID from the pools list) [136] .
- `GET /yields/lsdRates` – Current rates for LSD (Liquid Staking Derivatives) across providers [137] (useful for staking yield comparisons).
- `GET /yields/perps` – Funding rates and open interest for perpetuals across exchanges (centralized and decentralized) [138] .
- There are also specialized endpoints: `.../poolsBorrow` for lending borrow rates [139] , `.../chartLendBorrow/{pool}` for borrow APY history [140] , etc., which cater to advanced yield analytics (lending/borrowing markets APY history).

- (For most use cases, the main `/yields/pools` and `/yields/chart` will be the primary ones for dashboard visualizations.)

- **Revenue and Fees:**
DefiLlama tracks protocol fees revenue and token incentives:

- `GET /api/summary/fees/{protocol}` – Historical fees/revenue for a given protocol [141] . This can be used to chart daily fees, etc.
- In the Pro API, fees data might also be integrated in the protocol details or separate endpoints for daily fee rankings. The `fees` category is a bit less documented, but the summary endpoint provides what is needed for one protocol.

- Additionally, there's `GET /api/overview/fees` or similar that might list multiple protocols (if provided, similar to DEX overview but for fees).

- **Token & Liquidity Analytics:**

- `GET /api/tokenProtocols/{symbol}` – Given a token symbol (e.g. `"usdt"`), lists how much of that token is held across all protocols [142] . This effectively shows where a token's liquidity is (the "Token Usage" page on DefiLlama). For example, USDT might be broken down: X in Uniswap, Y in Aave, Z in Curve, etc.

- `GET /api/historicalLiquidity/{token}` – Historical available liquidity for swapping a given token on DEXes [143]. This could show, for instance, how liquidity for USDT or a smaller token has grown or shrunk over time on DEXs.
- **Prices:** DefiLlama also has price endpoints (especially for tokens not listed on major CEXes):
  - `GET /coins/prices/current/{coins}` – Current prices for tokens by their chain and address [144]. You pass a comma-separated list like `ethereum:0xabc...,bsc:0xdef...` and it returns current price (sourced from DEX data if needed).
  - `GET /coins/prices/historical/{timestamp}/{coins}` – Price of given tokens at a specific timestamp [145].
  - `GET /coins/chart/{coins}?start=X&end=Y&span=N&period=...` – **Multi-token price chart** [146]. This returns price series for multiple tokens between given times, ideal for plotting relative performance. You can specify how many data points (`span`) and the interval (`period`, e.g. 1h or 1d). This is very useful: with one call you can get, for example, daily prices for 5 different tokens over the last year [147].
  - `GET /coins/percentage/{coins}?period=...` – Percentage change in price over a period [148]. This directly gives the percent change (and possibly the base and final prices) for specified tokens over a given period (like 24h, 7d, etc). If you just need the % change number to display, this saves calculating it manually [149].
  - `GET /coins/prices/first/{coins}` – The first price record DefiLlama has for each token (i.e., when it was first tracked) [150].
- **Other Data:**

- **Bridges:** Endpoints under `/bridges` – e.g. `GET /bridges` (list all bridge protocols with recent volume) [151], `GET /bridge/{id}` (details of a specific bridge's volume, maybe liquidity) [152], `GET /bridgevolume/{chain}` (historical bridge volume overall or per chain) [153], `GET /bridgedaystats/{timestamp}/{chain}` (24h breakdown of bridge volume by token) [154], `GET /transactions/{id}` (transactions through a bridge within a time range) [155]. If your dashboard covers cross-chain movement, these are relevant.
- **DEXes (Volumes):** Similar to Dune's DEX endpoints, DefiLlama has: `GET /api/overview/dexs` (all DEXes summary) [156], `GET /api/overview/dexs/{chain}` (DEX volume on one chain) [33], `GET /api/summary/dexs/{protocol}` (historical volume for one DEX) [157]. For **derivatives** DEXes (perpetuals/options): `GET /api/overview/derivatives` [158] and `GET /api/summary/derivatives/{protocol}` [159] for perp exchanges.
- **Options:** `GET /api/overview/options` and `/options/{chain}` – volume stats for options protocols [160] [161], plus `/summary/options/{protocol}` for specific platform's option volume [162].
- **NFTs/Hacks:** `GET /api/hacks` – List of all major hacks/exploits (with amounts, dates) [163]. `GET /api/raises` – List of fundraising rounds (projects raising money, amounts) [164]. `GET /api/treasuries` – Data on protocol treasuries (how much assets protocols hold) [165]. `GET /api/entities` – List of entities (like venture funds or organizations) and their related protocols [166].
- **Emissions & Unlocks:** `GET /api/emissions` – List of tokens that have emissions/unlock schedules [167]. `GET /api/emission/{protocol}` – Unlock schedule for a specific token (e.g., when investor tokens unlock, how much) [168]. This is great for charts of upcoming token unlocks.

As you can see, the DefiLlama Pro API essentially opens up **every dashboard** you see on defillama.com for API access. Pro access "unlocks" data categories like *unlocks, active users, token liquidity, raises* that were not available in the free API [169].

Authentication: DefiLlama's Pro API uses a key that is included in the request URL. The base endpoint is `https://pro-api.llama.fi` and the **API key is included as part of the path**. According to the docs, the format is:

```
https://pro-api.llama.fi/<YOUR_API_KEY>/<endpoint_path>
```

For example, if your key is `abc123...`, a request would look like:

```
curl "https://pro-api.llama.fi/abc123/api/tvl/uniswap"
```

DefiLlama's web docs allow you to input your key which then automatically injects it into the interactive calls [170]. Alternatively, you might be able to use a header like `Authorization: Bearer <key>` (or a query param) but the official method given is the URL path injection. Keep your key secret – do not expose it on the frontend. In a Replit backend, you can store it and prepend it to all request URLs.

Rate Limiting: The Pro plan allows **1000 requests/minute** and up to **1,000,000 calls per month** [111]. This is quite generous. Still, for heavy data (like fetching all pools which could be thousands of entries), be mindful of rate limits if you refresh data often. The **Open (free) API** by contrast is 10-200/minute [171], so Pro is a big upgrade. If you approach the monthly cap, DefiLlama provides a usage page and you can even set up alerts for call consumption [172] [173]. Typically, each endpoint call counts as 1 credit regardless of payload size.

Response Format & Data Schema: All DefiLlama responses are JSON. Common patterns: - For list endpoints (e.g., `/api/protocols`, `/stablecoins/stablecoins`), you get an **array of objects**, each object with fields like `id` (slug), `name`, and relevant metrics (TVL, or supply, etc). - For historical data endpoints, you often get an object with a `chart` or `timestamps` and `values` array. E.g., a TVL history might look like: `{"tvl": [ [timestamp1, value1], [timestamp2, value2], ... ] }` or sometimes an object with `date: value` pairs. DefiLlama tends to use Unix timestamps (seconds) for time. - For detailed protocol or pool endpoints, you get structured JSON possibly with nested fields. E.g., `protocol/{id}` might have `{ "name":..., "tvl": {...}, "chart": [...]}, "tokens": {...} }`. - Field units: TVL, volume, etc. are usually in USD (as numbers). Percentages might be given as actual percentages (e.g., `apy: 5.23` meaning 5.23%). Token amounts might appear in raw units or USD depending on context. - Nulls vs zeros: If a protocol doesn't have data for a category, you might see `null`. E.g., a protocol with no revenue data might have `revenue: null`.

DefiLlama's API returns a **lot** of data; it may need post-processing for visualization. For example, to recreate the "Top Protocols" table, you'd use `/api/protocols` and sort by `tvl`. To plot a chain's TVL over time, use `/api/v2/historicalChainTvl/{chain}` which returns a time series. For active users over time, you might combine `activeUsers` endpoints.

Example – Pulling TVL and Plotting: Suppose you want to plot the TVL of Uniswap on an area chart. You would: 1. Call `GET https://pro-api.llama.fi/<KEY>/api/protocol/uniswap`. The response will include Uniswap's TVL history by chain. For example:

```
{
  "name": "Uniswap",
  "slug": "uniswap",
  "tvl": {
    "total": 40000000,
    "chart": [
      [1698796800, 35000000],
      [1698883200, 36000000],
      ...
    ]
  },
  "chainTvls": {
    "Ethereum": { "tvl": 30000000, "chart": [ ... ] },
    "Polygon": { "tvl": 10000000, "chart": [ ... ] }
  },
  ...
}
```

Here, `chart` is an array of `[timestamp, tvl]`. You can take the `tvl.chart` data to plot the total TVL over time. (Or stack by chain using `chainTvls` data). 2. If the API didn't provide a combined chart, an alternative is to use `GET /api/tvl/uniswap` for current value and `GET /api/summary/fees/uniswap` for daily fee chart, etc., depending on what you want to visualize.

Pagination: Most DefiLlama endpoints return all data in one go (no pagination needed). For instance, `/yields/pools` could return thousands of pools in one response. That might be heavy for the client, so consider filtering if possible (some endpoints allow query params to filter by chain or protocol). When dealing with large data sets (like all pools or all bridges transactions), you may need to handle large JSON in your Replit environment – possibly store and cache results to avoid repeatedly pulling huge lists.

Using DefiLlama Data in Dashboards: With Pro access, you can track metrics like: - **TVL charts** for each protocol or chain (growth over time). - **Comparison of protocols** by TVL or revenue (fetch multiple and compare fields). - **Token distribution** (e.g., use `tokenProtocols` to show where a particular stablecoin is allocated). - **Yield rankings** (sort `/yields/pools` by APY to list top yields). - **Upcoming unlocks** (from `emission/{id}` – you could create a timeline of token unlock events). - **DeFi vs CeFi** – e.g., use CoinGecko global data vs DefiLlama DeFi data to show DeFi share of total crypto market.

Building the Replit Dashboard (Multi-Line Chart Example)

Finally, to **fetch and visualize data** in a Replit dashboard, you will combine these APIs appropriately:

1. **Choosing the Right API for the Job:**

2. For **price time-series** (especially multiple tokens), you have a few options:
    - **CoinGecko**: Use `/coins/{id}/market_chart` or `/market_chart/range` for each token. This is straightforward and gives you price vs time for each coin in USD. You'll need to merge the data by timestamps for a multi-line chart. (Coins might have slightly different timestamp sets; align by day or hour as needed.)
    - **DefiLlama**: Use `GET /coins/chart/{coins}` to fetch multiple tokens in one call [146]. This is very convenient – you provide a list of token addresses (or use the special `coingecko:bitcoin` syntax for some) and a time range, and it returns synchronized price series for all. For example, you could get daily prices of BTC, ETH, and SOL for the last 90 days in one response. The result will likely have a structure like `{ "coins": { "<coin1>": [[t1, price1], ...], "<coin2>": [...], ... } }`. You can then plot each coin's series. DefiLlama even has `percentage` endpoint to directly get percentage changes over a period [148], but for a chart over time, you want the actual series.
    - **Velo**: If you wanted exchange-specific pricing or more granular data (1-min resolution, etc.), Velo's `/rows` with `type=spot` can provide that. For example, you could pull minute-by-minute prices for BTC from Binance and ETH from Coinbase. However, combining multiple assets might require separate calls or careful parameter use (Velo's `coins` param allows multiple symbols, but it will return separate CSV lines per asset/exchange). This is more advanced if you need high-frequency data or cross-exchange comparisons.
3. For **percentage change lines** (like a normalized chart starting all lines at 0% at t0): fetch price series as above, then calculate percent change relative to the starting point for each series in your code. Many chart libraries can do this if you input the raw prices and set a baseline.

4. For **other metrics**:

    - If you want a multi-line chart of, say, TVL of several protocols over time, use DefiLlama's `/api/protocol/{id}` for each (or find if there's a multi-protocol endpoint). You'll get each protocol's TVL chart, then you can overlay them.
    - For **active users over time** for multiple projects, you might create a Dune query returning those metrics and use Dune's API to feed the chart.

5. **Visualization Libraries:** In a Replit dashboard (likely a web app or Jupyter notebook), you can use libraries like:

6. **Plotly** or **Matplotlib** (Python) if it's a backend script generating a graph.
7. **Chart.js**, **Recharts**, or **D3.js** if you're building a front-end web dashboard in Replit. These can consume JSON data via an API route.

8. **Mermaid or other simple charting** for quick line charts in certain environments. For a dark-themed multi-line chart (as in the screenshot), a popular choice is Chart.js with a dark theme or Plotly with template="plotly_dark".

9. **Data Refresh and Limits:** All these Pro APIs allow frequent updates. You can schedule your Replit (or use a loop with `setInterval` in JS or `time.sleep` in Python) to fetch new data periodically. Just respect rate limits:

10. If updating every minute, 60 calls/hour per endpoint is fine under Pro limits.

11. If pulling large histories, do it once and then incremental updates (e.g., get latest price or TVL point rather than full history each time).
12. Use caching where possible (Replit's filesystem or in-memory) to avoid redundant calls.
13. **Attribution:** All these services (especially CoinGecko and DefiLlama) request attribution if data is publicly displayed. Since this is an internal dashboard on Replit, you may not need to show logos, but if shared, note "Data from CoinGecko, DefiLlama" etc., as a courtesy.

Example Implementation Outline: Suppose we want to recreate a chart of price % change over time for Bitcoin, Ethereum, and Solana over the past 30 days: - **Data Fetching:** Use DefiLlama's multi-chart endpoint. For BTC and ETH which are base layer coins, DefiLlama might allow a special identifier (like `coingecko:bitcoin` or an internal id). For demonstration, let's assume we use contract addresses for wrapped versions (just as an example, WBTC, WETH, etc., or the coingecko: prefix):

```python
import requests
API_KEY = "YOUR_LLAMA_KEY"
coins_param = "coingecko:bitcoin,coingecko:ethereum,coingecko:solana"
url = f"https://pro-api.llama.fi/{API_KEY}/coins/chart/{coins_param}?
start=1689638400&end=1692230400&period=1d"
data = requests.get(url).json()
```

This might return data like:

```
{
  "coins": {
    "bitcoin": [[1689638400, 30000], [1689724800, 31000], ...],
    "ethereum": [[1689638400, 2000], [1689724800, 2100], ...],
    "solana": [[1689638400, 50], [1689724800, 55], ...]
  }
}
```

Then, in Python, calculate percentage changes relative to the first value of each series:

```python
pct_data = {}
for coin, series in data["coins"].items():
    prices = [p for ts, p in series]
    start = prices[0]
    pct_change_series = [(p - start) / start * 100 for p in prices]
    pct_data[coin] = pct_change_series
```

Now `pct_data` holds arrays of percent changes for each coin corresponding to each date index. - **Plotting:** Using Plotly for example:

```
import plotly.graph_objs as go
import plotly.io as pio
pio.templates.default = "plotly_dark"
dates = [ts for ts, p in data["coins"]["bitcoin"]]
fig = go.Figure()
for coin, pct_series in pct_data.items():
    fig.add_trace(go.Scatter(x=dates, y=pct_series, mode='lines', name=coin))
fig.update_layout(title="30d Price Change (%)", yaxis_title="% Change",
xaxis_title="Date")
fig.show()
```

This would render a dark-themed line chart with three lines for BTC, ETH, SOL showing their percentage change from Day 0.

- **Alternative using CoinGecko:** If not using DefiLlama, you'd call CoinGecko's `/market_chart` for each coin (with `vs_currency=usd&days=30`). Merge on dates (CoinGecko gives daily data by default if days> max timeframe). Calculate percent change similarly, then plot.

- **Other Dashboard Elements:** You can similarly use these APIs to populate tables (e.g., top 10 protocols by TVL using DefiLlama data), bar charts (e.g., revenue last week for various protocols), or text indicators (e.g., current price, 24h change from CoinGecko, updated every minute). The key is to identify which service has the *latest* or *historical* data you need:

- **Latest prices/market caps**: CoinGecko `/coins/markets` or DefiLlama `/coins/prices/current`.
- **Historical trends**: DefiLlama for TVL, CoinGecko for price, Dune for custom on-chain metrics, Velo for intraday price/funding.
- **Special metrics**: e.g., "active users" might require Dune (if you write a query for on-chain active addresses) or DefiLlama's `activeUsers` if provided.

In summary, by leveraging **Velo for granular market data**, **CoinGecko for broad market and price info**, **DefiLlama for DeFi-specific metrics**, and **Dune for custom analytics**, you can fetch a wealth of data and assemble a rich Replit dashboard. Use the authentication methods as described for each, adhere to rate limits (which are high on Pro plans), and parse the JSON/CSV responses to feed your visualizations. Always refer to official docs for field definitions and update your API calls if the providers make changes (since these platforms evolve quickly). With the above endpoints and examples, you should be able to recreate dynamic charts and dashboards similar to the example screenshot – for instance, a dark-themed multi-line chart showing multiple tokens' performance over time can be achieved by combining CoinGecko or DefiLlama price endpoints with a charting library [103] [148] .

Sources:

- Velo API Documentation – Endpoints and usage [4] [174] [16]
- Velo News/WebSocket API [12] [14]
- Dune Analytics API Docs – Overview and Preset Endpoints [175] [31]
- Dune API Authentication and Examples [39] [28]

- CoinGecko API Docs – Endpoint Categories and Authentication [51] [42]
- CoinGecko Pro API Guide – Using API key and on-chain data [49] [95]
- DefiLlama Pro API – Features and Endpoints (via DefiLlama docs and schema) [169] [112] [114]
- DefiLlama API Schema (community compiled) – List of Pro endpoints [176] [164] [146]
- DefiLlama Pricing and Limits [111] [110]

---

[1] [3] [24] Intro | Velo
https://docs.velo.xyz/

[2] [4] [5] [6] [7] [10] [11] [16] [17] [174] HTTP | Velo
https://docs.velo.xyz/api/http

[8] [9] [18] [19] [20] [21] [22] [23] Columns | Velo
https://docs.velo.xyz/api/columns

[12] [13] [14] [15] News API | Velo
https://docs.velo.xyz/api/news-api

[25] [27] [35] [36] [38] [175] Dune API Overview - Dune Docs
https://docs.dune.com/api-reference/overview/introduction

[26] [39] [40] [43] [47] [48] Authentication - Dune Docs
https://docs.dune.com/api-reference/overview/authentication

[28] [29] [30] [31] [37] DEX Pair Stats - Dune Docs
https://docs.dune.com/api-reference/dex/endpoint/dex_pair

[32] [33] [34] [112] [113] [114] [115] [116] [117] [118] [119] [120] [121] [122] [123] [124] [125] [126] [127] [128] [129] [130] [131] [132] [133] [134] [135] [136] [137] [138] [139] [140] [141] [142] [143] [144] [145] [146] [147] [148] [149] [150] [151] [152] [153] [154] [155] [156] [157] [158] [159] [160] [161] [162] [163] [164] [165] [166] [167] [168] [176] Schema | REI Crypto MCP Server | Glama
https://glama.ai/mcp/servers/@0xReisearch/crypto-mcp-beta/schema

[41] [42] [95] [100] [101] [102] [103] [104] [105] [106] Authentication (Pro API)
https://docs.coingecko.com/reference/authentication

[44] Rate Limits - Dune Docs
https://docs.dune.com/api-reference/overview/rate-limits

[45] FAQ - Dune Docs
https://docs.dune.com/api-reference/overview/faq

[46] Get Latest Query Result - Dune Docs
https://docs.dune.com/api-reference/executions/endpoint/get-query-result

[49] [172] [173] Setting Up Your API Key
https://docs.coingecko.com/reference/setting-up-your-api-key

[50] [51] [52] [53] [54] [55] [56] [57] [58] [59] [60] [61] [62] [63] [64] [65] [66] [67] [68] [69] [70] [71] [72] [73] [74] [75] [76] [77] [78] [79] [80] [81] [82] [83] [84] [85] [86] [87] [88] [89] [90] [91] [92] [93] [94] [96] [97] [98] [99] [107] [108] Endpoint Overview
https://docs.coingecko.com/reference/endpoint-overview

[109] How can I get a list of the 300 first coins of coingecko API by ...
https://stackoverflow.com/questions/63075151/how-can-i-get-a-list-of-the-300-first-coins-of-coingecko-api-by-marketcap

[110] [111] [169] [171] Pricing | DefiLlama
https://docs.llama.fi/pro-api

[170] Scalar API Reference
https://api-docs.defillama.com/