What If Execution Prices Were Predictable?

Adam Davis Cuculich

1 Introduction

We begin by setting precise definitions, which will guide the rest of our exploration into predicting the execution price of token swaps. Ultimately, we show that two distinct sources of error—vendor error and slippage—must both be addressed in order to produce realistic, risk-buffered execution prices.

2 Foundations

2.1 Automated Market Makers as Functions

A swap on an AMM can be modeled as a function from input to output:

$$\text{amt}_{\text{out}} = f(\text{amt}_{\text{in}}, S),$$

where S denotes on-chain state (reserves, fees, etc.).

2.2 The Spot Price

Let $f(\text{amt}_{\text{in}}, S)$ map input to output at state S. The marginal *output per unit input* is

$$\text{marginal\_amt}_{\text{out}}(S) \;=\; \frac{\partial f(\text{amt}_{\text{in}}, S)}{\partial \text{amt}_{\text{in}}}\Big|_{\text{amt}_{\text{in}}=0}.$$

The marginal (spot) price is then obtained by taking its reciprocal:

$$\text{marginal\_price}_{\text{out token}} \;=\; \frac{1}{\text{marginal\_amt}_{\text{out}}(S)}.$$

2.3 The Effective Price of a Finite Trade

For a finite trade, the *effective price of the output token in terms of the input token* is

$$\text{price (out in input)} \;=\; \frac{\text{amt}_{\text{in}}}{\text{amt}_{\text{out}}}.$$

For sufficiently small trades ($\varepsilon \to 0$), the effective price converges to the marginal price:

$$\lim_{\varepsilon \to 0} \frac{\varepsilon}{f(\varepsilon, S)} \;=\; \frac{1}{\text{marginal\_amt}_{\text{out}}(S)}.$$

2.4 Effective Price for Multi-Hop Routes

A K-hop route is a composition of AMM maps $f^{(1)}, \ldots, f^{(K)}$ with (possibly) hop-specific states $S^{(1)}, \ldots, S^{(K)}$. Define

$$F(\text{amt}_{\text{in}}, S) \;=\; f^{(K)}\big(\, f^{(K-1)}(\cdots f^{(1)}(\text{amt}_{\text{in}}, S^{(1)}) \cdots, S^{(K-1)}), S^{(K)}\big),$$

so the final output is $\text{amt}_{\text{out}} = F(\text{amt}_{\text{in}}, S)$. The effective price of the output token in terms of the input token is *unchanged in form*:

$$\text{price (out in input)} \;=\; \frac{\text{amt}_{\text{in}}}{F(\text{amt}_{\text{in}}, S)}.$$

This generalization justifies incorporating DEX aggregators—which search multi-hop routes—as a necessary (though not sufficient) component of an accurate pricing oracle.

2.5 Why Execution Differs From a Quote

Even though the effective price is well-defined, the realized execution often deviates from a quote due to two contributors, which we model as approximately additive:

$$\Delta \;\approx\; \Delta_{\text{vendor}} \;+\; \Delta_{\text{slippage}},$$

where Δ is the fractional output error relative to the quote.

1. **Vendor error.** Quote-model bias/optimism, fee/rounding treatments, API or routing approximations can make the quoted output misaligned with ground truth at quote time.

2. **Slippage.** State drift $S \to S'$ between quote and execution (later block) changes pool reserves; the magnitude of the drift depends on (i) the number of intervening transactions, (ii) their sizes, and (iii) trade direction (e.g., USDC $\to$ ETH vs. ETH $\to$ USDC).

2.6 Accurate Pricing Requires Both Components

Because both vendor error and slippage contribute to divergence between quoted and actual execution prices, **a realistic pricing mechanism must explicitly account for both**.

3 Solution Design Principles

3.1 Constraints

The solution must be lightweight, low-latency, and inexpensive. In particular, it should *not* rely on simulating actual transactions before they are executed, since simulations are costly in both time and computation. Instead, the approach should exploit pre-computation and pre-modeling, keeping online adjustments minimal.

3.2 Assumptions

1. **Local, route-specific prices.** We seek accurate execution prices for a *user's specific trade and route*, not a global TVL-weighted average across venues.
2. **Additivity (approximate).** The total discrepancy is modeled as approximately additive, $\Delta \approx \Delta_{\text{vendor}} + \Delta_{\text{slippage}}$.

3.3 Price Adjustment

The goal is to correct quoted outputs with predicted adjustments.

Define the total fractional error

$$\Delta \;=\; \frac{\text{amt}_{\text{out,actual}} - \text{amt}_{\text{out,quote}}}{\text{amt}_{\text{out,quote}}} \quad (\Delta < 0 \text{ worse than quote; } \Delta > 0 \text{ better}).$$

To use Δ operationally, we *decompose* it into components:

$$\Delta \;=\; \Delta_{\text{vendor}} \;+\; \Delta_{\text{slippage}} \;+\; \varepsilon_{\text{int}},$$

where Δ_{vendor} captures misalignment at quote time, Δ_{slippage} captures state drift between quote and execution, and ε_{int} is a small interaction/residual term. For the remainder, we assume $\varepsilon_{\text{int}} \approx 0$ (additive model), i.e.

$$\Delta \;\approx\; \Delta_{\text{vendor}} \;+\; \Delta_{\text{slippage}}.$$

The adjusted output estimate is

$$\widehat{\text{amt}}_{\text{out}} \;=\; \text{amt}_{\text{out,quote}} \left(1 + \widehat{\Delta}_{\text{vendor}} + \widehat{\Delta}_{\text{slippage}}\right).$$

Notation: a hat $\widehat{(\cdot)}$ denotes a model estimate/prediction.

Since the effective price of a trade is

$$\text{price (out in input)} \;=\; \frac{\text{amt}_{\text{in}}}{\text{amt}_{\text{out}}},$$

substituting $\widehat{\text{amt}}_{\text{out}}$ yields an adjusted effective price that reflects predicted vendor error and slippage.

4 A Solution for Estimating Vendor Error

Build a Database for Supervised Learning. The first step is to assemble a dataset that captures the discrepancy between quoted and actual execution prices. This can be approached as follows:

1. **Identify the trade set.** Define a historically-informed set of trades we care about. These can be drawn from Relay's own trade history or from broader market data.

2. **Select input amounts.** For each trade, define a range of input sizes (e.g., \$1, \$10, \$100, \$1,000, \$10,000), also informed by historical patterns.

3. **Collect quotes.** At a fixed cadence (e.g., daily), request quotes from the aggregators of interest across all trades and input sizes.

4. **Simulate execution.** For each quote, simulate execution on a fork at the same block to obtain the ground-truth output.

5. **Record discrepancies.** Store both the quoted output and the simulated output, computing the difference as a labeled error term.

4.1 Dataset Schema

The following schema captures the minimal set of fields required for modeling vendor error:

- **Must-haves (MVP core):** `ts`, `chain`, `vendor_id`, `pair`, `amount_in_token`, `amount_in_usd`, `quote_out_token`, `quote_out_usd`, `quote_gas_usd`, `spot_price_in`, `spot_price_out`, `gas_price`, `actual_out_token`, `actual_out_usd`, `actual_gas_usd`, `delta_frac`.
- **Nice-to-haves (optional):** `hop_count`, `min_pool_tvl_usd`.

Here, the label of interest is

$$\Delta_{\text{vendor}} = \frac{\text{actual\_out\_token} - \text{quote\_out\_token}}{\text{quote\_out\_token}}.$$

4.2 Supervised Learning For Vendor Error

With this dataset, we can learn a mapping

$$(\text{chain}, \text{vendor}, \text{pair}, \text{amount\_in}, \text{hop\_count}) \longmapsto \Delta_{\text{vendor}}.$$

As the dataset grows, a neural network can be trained to predict Δ_{vendor} for new quotes using the relevant features. Because vendor error distributions evolve over time, it is important to (1) **retrain models on a regular schedule**, and (2) **apply a recency bias**, giving higher weight to recent observations.

4.3 Beyond Price Estimation

This dataset not only enables more accurate execution price prediction, but can also inform aggregator selection. For example, if a particular aggregator is consistently found to provide optimistic or inaccurate quotes, then given our prediction of its error, we may choose to route the trade through a different aggregator that offers a more reliable outcome.

5 A Solution for Estimating Slippage

We previously defined slippage as the difference between the quoted and executed prices caused by intervening market activity between quote time and execution. To isolate slippage, we must first remove the contribution of vendor error. In practice, this means constructing a dataset that ideally only contains trades where the vendor quote was accurate (i.e., $\delta_{\text{vendor}} \approx 0$). With vendor error filtered out, the resulting discrepancies between quoted and actual outputs can be attributed to slippage. Such a dataset could be assembled gradually from real, ground-truth Relay transactions.

5.1 Feature Choice and the Causes of Slippage

The choice of features used for prediction is particularly important for slippage prediction. To reason about which features may be most informative, we consider two complementary categories: (1) intentional slippage reduction, and (2) historically-informed estimation.

5.1.1 Slippage Reduction Strategies

Certain parameters of a trade can be controlled in order to reduce the likelihood or magnitude of slippage. Examples include:

1. **Liquidity constraints.** Restrict routing to pools with a minimum total value locked (TVL). Higher-liquidity pools can absorb trades with less price impact.

2. **Like-asset routes.** Restrict routing to trades between highly correlated assets (e.g., ETH $\rightarrow$ WETH, or USDC $\rightarrow$ DAI). Such trades often utilize specialized invariant curves (e.g., Curve's 3Pool) designed to minimize price impact. Some DEX aggregators allow for this customization, as a quote request parameter.

3. **Hop count minimization.** Prefer routes with fewer swaps. Slippage occurs at the AMM level; each additional hop introduces another opportunity for adverse price movement. Although slippage can, in principle, be positive, DEX aggregators typically capture positive slippage as profit. Thus, from the user's perspective, additional hops increase downside risk without compensating upside.

5.1.2 Historically-Informed Estimation

Beyond proactive route design, slippage can also be estimated from historical data:

- **Mapping features to slippage deltas.** Construct a dataset mapping trade descriptors (pair, chain, vendor, route hop count, input size) to observed slippage deltas.

- **Inclusion of time-series features.** Since slippage is driven by market activity between quote and execution, features capturing trade momentum or transaction density in the preceding blocks may be valuable.

5.2 Supervised Learning for Slippage

As with vendor error, the goal is to learn a supervised mapping:

$$(\text{pair}, \text{chain}, \text{vendor}, \text{amount\_in}, \text{hop\_count}, \ldots) \longmapsto \Delta_{\text{slippage}}.$$

Unlike vendor error, it is not clear that older data becomes less relevant. Thus, recency bias may be unnecessary. Instead, the dataset should emphasize breadth (many route configurations) and **temporal context** (features capturing short-term trade momentum).

6 Why This Solution Meets the Constraints

- **Low latency.** Models are trained offline; online work is a single forward pass on a small feature vector plus a few arithmetic operations—no on-path simulation.
- **Any trade size.** Inputs come from DEX aggregators, whose quotes already account for route choice and size-dependent impact; the adjustment operates directly on these size-aware quotes.
- **Low cost.** Inference is constant-time and can run on commodity CPUs; data collection and training are batch/offline.
- **Risk-buffered output (optional).** Use a lower quantile prediction for the corrections to provide downside protection.

7 Conclusion: A Unified Pricing Mechanism

By combining predicted vendor error and predicted slippage, we can produce risk-buffered execution price estimates. The adjustment equation integrates both error sources, yielding a realistic price that can be served in production. This unified approach ensures that vendors can be selected not just on optimistic quotes, but on adjusted outcomes that reflect true expected execution.