Doppler: A liquidity bootstrapping ecosystem

Jan 2024

Austin Adams
austin@whetstone.cc

Matt Czernik
matt@whetstone.cc

Clement Lakhal
clement@whetstone.cc

Kaden Zipfel
kadenzipfel@gmail.com

Abstract

Doppler is a liquidity bootstrapping ecosystem that facilitates liquidity provision by introducing a new primitive: a dutch-auction dynamic bonding curve, which is used to source initial two-sided liquidity. This mechanism is a self-executing and non-custodial Uniswap v4 hook designed for automated initial price discovery for arbitrary assets. Additionally, Doppler provides infrastructure to allow ecosystems to go from token deployment to an arbitrary liquid generalized automated market maker without user intervention.

1 Introduction

The rise of automated market makers (AMMs) has fundamentally transformed digital asset trading. Protocols, such as the Uniswap Protocol [4], have resulted in a shift towards automated pricing algorithms. This shift has created an exponential increase in the number of traded onchain assets, with more than 910,000[1] unique traded assets created as of writing. Despite this, token projects continue to struggle to bootstrap liquidity, and oftentimes must outsource to automated liquidity management protocols or onchain market making desks, which generally take a sizeable portion of the provided assets.

However, one benefit of blockchain-based ecosystems is that they allow for programmable markets. Just as the Uniswap Protocol is designed to facilitate trading through an $xy = k$ model, a similar model could be created to bootstrap liquidity. Indeed, we have already seen such projects on both Solana, Tron, and Ethereum.

Traditionally, the problem with creating new market structures is the difficulty of creating customized AMM protocols, which require expertise and significant time and capital investment. However, the forthcoming Uniswap v4 [2] improves the customization and integrator experience of AMMs by introducing hooks, which can execute developer-defined logic at specific points in the transaction's lifecycle. While Uniswap v3 [4] focused on optimizing the most common trading flow, Uniswap v4 allows custom market structure models for specialized tasks.

2 Background

The proliferation of AMMs has created new challenges in market structure and liquidity formation. Traditional AMM designs require significant upfront capital commitments from token projects. For instance, Uniswap v2 [3] requires two-sided liquidity provision, creating both financial and technical barriers to market formation. Projects must not only secure substantial capital for both sides of the pool but also determine appropriate initial pricing which carries significant risk of capital loss through mispricing. While projects such as Uniswap v3 [4] have utilized single-sided liquidity, which requires only one asset, token projects must still take an initial stance on the asset's price.

The challenge of initial price discovery is particularly problematic for initial liquidity provision. Generally, because of arbitrage forces, the pool already liquid and is in-line with the market price, so adding more liquidity does not create the potential for more losses. However, initial liquidity provision does not have this guarantee. Projects face a difficult trade-off: setting initial prices too low results in immediate capital loss through arbitrage, while setting prices too high inhibits trading activity entirely.

A fundamental nature of permissionless systems, like email or TCP, is the vast amount of value-less user generated content. This type of market structure is formalized in Oh [10], which models non-fungible tokens (NFTs) as "digital Veblen goods." The implication of this model is that the purchasing of an asset greatly increases likelihood that a product is valuable.

This market dynamic is well-supported by static bonding curves implemented in projects like friendtech. Bonding curves start assets with cheap prices with the next trade being marginally more expensive than the previous. However, these protocols can run into significant issues from programmatic bots, who buy these cheap assets and instantly sell if the price increases. This strategy is very low risk, because the bonding curves are static. The initial buyers of the tokens cannot lose any money. The price either goes up as new users buy, allowing the initial buyers to sell and collect a profit, or (if no one bought) they sell at the original price they paid (due to the static bonding curve). Single-sided liquidity bootstrapping from Intuitive Launch Pool functions similarly to this model, utilizing concentrated liquidity math as the bonding curve.

Doppler provides a dutch-auction dynamic bonding curves liquidity bootstrapping hook and a token factory to create ERC20s with known bytecode. This allows ecosystems to go from token deployment to liquid generalized automated market maker, like Uniswap v2 or Uniswap v4, without user intervention.

- *Dutch-auction Dynamic Bonding Curves*: Price discovery of assets is facilitated through a dutch-auction dynamic bonding curve. This mechanism is built on top of Uniswap v4, facilitating automated price discovery and auctioning

[1] Source: https://dune.com/queries/3891972

without intervention. Additionally, user experience should be identical to and supported by the current swapping experience.

- *Token contract Factory*: The ecosystem utilizes a token contract factory to deploy ERC20s with known bytecode. This bytecode removes many of the malicious implementations prevalent in the EVM ecosystems and allows 3rd party interfaces to quickly check bytecode. Additionally, this factory and hook combine to enforce invariants on token trading.
- *Direct to generalized liquidity*: Once the required tokens are bootstrapped to create a liquid pool, the Uniswap v4 hook contract deposits the funds into a generalized AMM without intervention. However, unlike other liquidity bootstrapping implementations, the underlying LP share tokens are not burned. In future versions, users can choose between Uniswap v2, a CFMM on Uniswap v4, or any arbitrary Uniswap Protocol pools.
- *Time-locked liquidity*: Liquidity tokens for the generalized liquidity pool are time-locked initially, but can be withdrawn at a later date. This is to ensure that a community around that token can optionally utilize these funds at a later date.

The following sections provide in-depth explanations of each part of the token deployment ecosystem.

3 Dutch-auction Dynamic Bonding Curves

Dutch-auction Dynamic Bonding Curves blend together two well-studied mechanisms in blockchain-based markets - dutch auctions and bonding curves.

3.1 Background

Dutch auctions are a well-understood auction format, and have been studied deeply for blockchain-based markets in pieces such as Frankie (2022) [7] and Moallemi (2024) [9]. Dutch auctions are descending price auctions, where the auction starts at a high price and decays until a market clearing price is found. They are generally utilized within onchain markets due to their efficient implementation and "shill-proof" properties [8]. The descending price structure of Dutch auctions naturally mitigates the information asymmetry challenges inherent in blockchain markets, where bid revelation carries both explicit costs through gas fees and implicit costs through information leakage. Dutch auctions provide market participants with clear decision boundaries, enabling more efficient price discovery while minimizing strategic gaming opportunities.

On the other hand, bonding curves are another widely used blockchain-based markets primitive, which attempts to mimic the laws of supply and demand. As previously stated, static bonding curves have significant issues around determining the optimal starting levels. Too low a starting price, and significant value is lost due to the instantaneous arbitrage. Additionally, this value is lost to programmatic users who will instantly dump on other users, causing losses for users who desire long-term exposure to that ecosystem. For context, instantaneous arbitrage has cost token deployers over +$100 million dollars on Ethereum mainnet over the last year[2]. Likewise, instantaneous arbitrage and MEV have cost more than $400m from users on pumpdotfun on Solana[3]. Too high and trades never occur, but this is heavily mitigated by the dutch auction mechanism.

3.2 Implementation

The dutch-auction dynamic bonding curve has two phases

- Rapid price decrease until an initial market clearing price is found
- Once a lower bound price is found, the expected price ramps up slowly using the dynamic bonding curve

The optimal price curve is shown below. The price of the marginal token rapidly decreases until users begin buying, and the bonding curve behavior begins. This market behavior encourages the starting price of the dutch auction to be higher than "the market price", limiting losses from instantaneous arbitrage.

It is important to note that if the price were to start "too low", the descending price auction would not occur, and the dynamic bonding curve behavior would begin. There is no requirement in this mechanism for the descending price behavior - it is simply encouraged to maximize eventual liquidity. However, a significant price movement would cause the bonding curves to shift upward in a process that will be described in a later section.

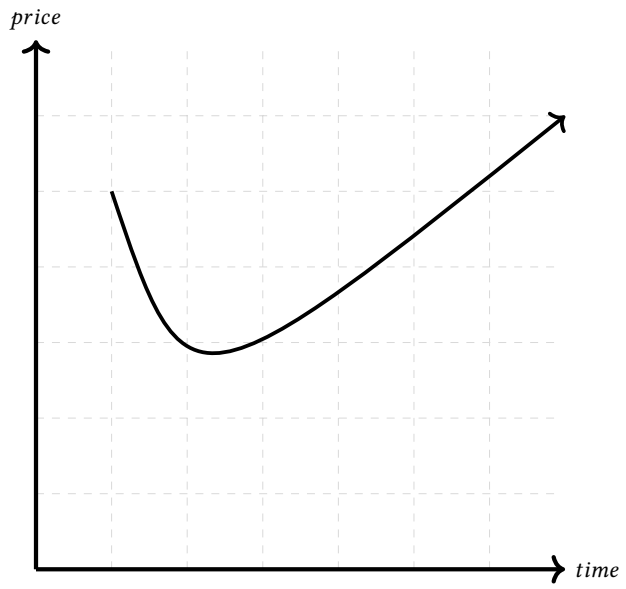

Figure 1: Optimal price curve

3.2.1 Liquidity Provision In order to protect against instantaneous arbitrage, `Doppler` streams liquidity into the pool at a configurable rate at each time. We define this liquidity at time t as λ_t. While the current implementation utilizes a constant streaming rate at epoch boundaries, this is only to simplify the code base.

The protocol tracks two critical liquidity metrics: the expected tokens sold (λ_t) and realized amount of "net-sold" tokens ($\hat{\lambda_t}$). The latter metric tracks tokens held outside the bonding curve system, providing a crucial measure of market absorption capacity. The

[2]Source: https://dune.com/queries/3682272/6193594

[3]Source: https://x.com/0xngmi/status/1823399442306801974

differential between these metrics serves as a primary input for the dynamic adjustment of price curves over time.

3.2.2 Dutch Auction The dutch auction handles the decrease in price of tokens within the system. To start, the beginning price and end price of the dutch auction behavior is configurable by an interface that submits the trade. Interfaces may adjust these parameters depending on the expected starting price, potential volatility of the token, or based on broader market conditions.

Similar to other dutch auction protocols like UniswapX [5] or 1inch Fusion [1], Doppler uses a step-wise decaying function. In practice, due to `tickSpacing` in Uniswap v4, the price may not actually decay until it has passed a `tickSpacing`.

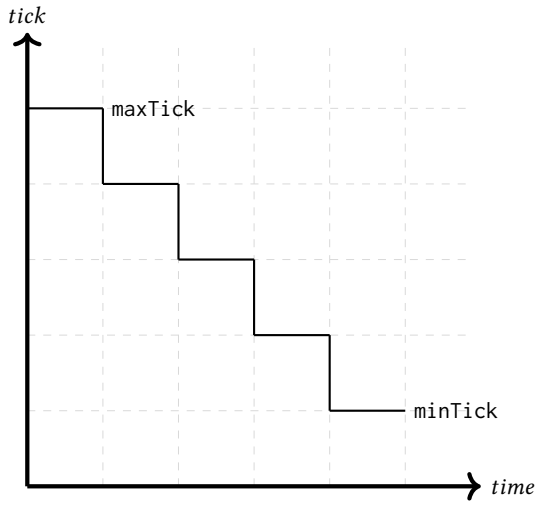

Figure 2: Target tick decay over time

Notice that the y-axis in Figure 2 says `tick`. Due to implementation details of Uniswap v4, it is easier to decay the price in log-space. This is implemented through a linear decay of the underlying ticks from a given `maxTick` to a given `minTick`, which are then mapped into the price using concentrated liquidity math.

Ideally, the decay should never reach `minTick`, as the bonding curve (described below) should take over price discovery. If the end of the sale time is reached and the required proceeds are not met, users are refunded a pro-rata share of the pool holdings if under the minimum threshold. Above this threshold will cause the liquidity to be sent to Uniswap v2 in the current design. The natural state of the pool is decaying to `minTick`, which will occur if no tokens are purchased.

To support price movements caused by both the dutch auction and the dynamic bonding curve, the hook contract keeps an `tickAccumulator` that aggregates tick adjustments from both mechanisms.

Since we are linearly dutch auctioning, we can define the max decrease as such

$$maxDelta = \frac{maxTick - minTick}{endingTime - startingTime} \cdot epochLength \tag{3.1}$$

At each curve re-balance, the hook calculates expected amount of tokens from the liquidity function, λ_t, sold from the previous epoch to the current and the realized amount sold, $\hat{\lambda_t}$. There are three states depending on the amount of tokens that are sold

$$tickDelta_t := \begin{cases} maxDelta & \hat{\lambda_t} - \lambda_t \leq 0 \\ \frac{maxDelta \cdot \hat{\lambda_t}}{\lambda_t} & 0 < \hat{\lambda_t} < \lambda_t \\ tickDelta_{u,t} & \hat{\lambda_t} \geq \lambda_t \end{cases} \tag{3.2}$$

Note that $tickDelta_{u,t}$ will be defined in the Bonding Curve section.

The `tickDelta` is added to the `tickAccumulator` during the re-balance of the curve, which is the accumulation of each `tickDelta` from start to time t.

$$tickAccumulator_t = \sum_{i=0}^{t} tickDelta_i \tag{3.3}$$

If the `tickDelta` is 0, then exactly the expected number of tokens were sold.

3.2.3 Dynamic Bonding Curves The dutch auction attempts to decay until the market finds the "market clearing" bonding curve. The decay from the dutch auction acts like an impulse downward, which is countered by buy pressure (which raises the price), while the upward target price of the pool is determined by current active bonding curve. We refer to the current active bonding curve as the bonding curve that the pool is checking its current price against.

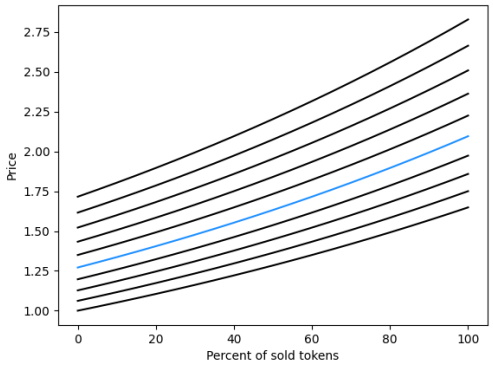

Figure 3: Dynamic Bonding Curves with γ = 500

First, we define the origin tick of the bonding curve, τ_t. To calculate this, Doppler takes the `startingTick` and adds the `tickAccumulator`. Note that the `startingTick` is either `minTick` or `maxTick` depending on if the sold token is `token0` or `token1`.

$$\tau_t = startingTick + tickAccumulator_t \tag{3.4}$$

Above you can see an example of a collection of bonding curves, with the current active one in blue. The current bonding curve $b_c(t)$ can be calculated at any time from the origin tick (τ_t), a growth parameter (γ), and the elapsed time (t).

$$b_c(t) = \gamma \cdot (t/t_{max}) + \tau_t \tag{3.5}$$

Notice that τ_t is also the bottom/start of the bonding curve and the slope of the bonding curve is given by γ. A higher γ means that the bonding curve is steeper. Using the Uniswap v4 concentrated liquidity math, the tick can be converted to price.

$$p(t) = 1.0001^{b_c(t)} \tag{3.6}$$

Checking the surrounding bonding curves is also trivial as they are a constant `tickSpacing` away. The current bonding curve is the position utilized by the pool for liquidity provision.

As time progresses in the pool, the current price must drift upward or downward, thus the desired bonding curve may change. To calculate when we should change the bonding curve, we can calculate the indifference curve between the current bonding curve against its two surrounding curves. Let τ_t be the current starting `tick` at time t. We can define the two boundary curves, $b_u(t)$ and $b_l(t)$

$$b_u(t) = \gamma \cdot (t/t_{max}) + (\tau_t + tickSpacing/2) \tag{3.7}$$

$$b_l(t) = \gamma \cdot (t/t_{max}) + (\tau_t - tickSpacing/2) \tag{3.8}$$

At the end of the epoch, if there was enough or extra tokens sold, we then want to increase the current target bonding curve. In this case, We calculate the current spot position of pool, i_c, against the expected position on the bonding curve.

$$tickDelta_{u,t} = i_c - (\gamma \cdot (t/t_{max}) + \tau_t) \tag{3.9}$$

We then adjust the `tickDelta` to be on a boundary defined by the `tickSpacing` of the pool. This adjustment functions similarly to the indifference curves defined above.

3.2.4 Position Strategy As previously described, `Doppler` streams liquidity into the pool according to a pre-described formula λ_t. At each time period, `Doppler` calculates the current amount of net-sold tokens, $\hat{\lambda_t}$ and the amount of tokens expected to be sold before the next epoch, λ_{t+1}. This amount of tokens is placed between the current tick, i_c and the $b_c(t+1)$, which is the expected price of the pool at the next epoch. If $(\lambda_{t+1} - \hat{\lambda_t}) \leq 0$, this position is skipped.

Next, `Doppler` calculates the expected amount of tokens sold between the next epoch λ_{t+1} and the epoch after that λ_{t+2}. This liquidity is placed from either the top of the previous position or the current tick, and then to the $\gamma + \tau_t$, which is the top of the current bonding curve.

3.2.5 Implementation Details In Uniswap v4, i_c is the current tick of the pool, determined through swapping. It is reasonable to ask if the pool bonding curve can be manipulated as i_c is the spot price of the pool.

Because the bonding curve is set in the `beforeSwap` hook in Uniswap v4, the hook is able to respond to manipulations during their execution (and cannot be censored). Because of this, a manipulator would lose funds from the shifting of the bonding curve, functioning as limit orders which are a strong mitigation to manipulations. Additionally, the portion of the bonding curve that allows selling of the tokens back may disjoint itself in a process described below.

On Ethereum mainnet (or any chain without strong censorship resistance[6]), it is possible for multiple slots in a row to be purchased by a manipulator (multi-block MEV). This actor could censor buy transactions until the price of the pool decreases, locking in a lower than market clearing price. A mitigation to this attack is that the price increase defined by the provided γ should always be greater than `maxDelta`. Additionally, a chain could be utilized with censorship resistance, meaning that that this censorship would always lose to in priority gas auction for a backrun. This is because the rebalances occur before the first trade of the epoch, meaning that manipulations would need to span blocks and epochs. In practice, extractive multi-block MEV is rare on Ethereum mainnet due to validators' incentives. This problem is not present on any chain with censorship resistance, like most Layer 2 protocols.

One key design decision is to keep both tokens in the pool fully liquid at all times. However, it is possible that the current bonding curve does not have enough quote token liquidity to support $b_c(t)$ from τ_t to i_c. This is likely from a misspecified starting price or rapid price appreciation (either manipulations or naturally).

If the pool is in this state, then we utilize two different $b_c(t)$ above and below the current i_c. The only way to move the i_c down would be selling tokens into the pool. In this state, we create a one `tickSpacing`-wide position at the price which supports a theoretical liquidation of the entire net-sold liquidity amount, $\hat{\lambda_t}$. This is to discourage run behavior and give a common clearing price to all users who wish to sell back their tokens.

4 Airlock and Factory Modules

The Doppler ecosystem implements a modular architecture centered around an airlock contract that serves as the primary entry point for system integration. This design enables seamless progression from token deployment to fully operational AMM markets while maintaining robust security guarantees.

4.1 Core Architecture

The airlock contract manages system progression through a "turnkey" mechanism that coordinates the sequential execution of module functions. This orchestration allows the system to automatically advance through deployment stages without requiring additional user intervention. The modular design philosophy, inspired by `Uniswap v4`'s hook architecture, separates core functionalities into distinct, immutable contracts while maintaining system flexibility.

The ecosystem comprises four primary module types that integrate with the airlock system:

- Token Factory
- Liquidity Factory
- Migration Factory
- Timelock Factory

The benefit of this system is the rapid creation of new user actions while the previous code remains immutable for existing developers. Users can additionally modify only one specific action without editing others, supporting customization without requiring the redeployment of the entire Protocol.

We will provide a quick overview of each factory.

4.1.1 Token Factory The token factory addresses a fundamental challenge in decentralized markets: the proliferation of potentially malicious token implementations. While the ERC20 standard defines basic token functionality, its minimalist specification leaves

considerable room for harmful implementations. Traditional approaches rely on heuristic checks to identify malicious contracts, creating an unsustainable arms race between security measures and exploitation techniques. By using a factory, swappers and integrators can trust that any arbitrary user-deployed contracts that emerge from the factory will meet certain standards.

In the near future, new modules that allow even more trustless customization will be created for the token factory, extending the possible use-cases for integrators. We note that the Airlock contract itself checks several invariants in the system to further mitigate possible attack vectors, but cannot reasonably check for every possible attack.

4.1.2 Liquidity Factory The liquidity factory creates and facilitates the entire process of generating the other token liquidity. While the core Doppler Protocol uses `Uniswap v4`, the liquidity hook is fully generalized to support the use of Uniswap v3 or any other AMM.

To interact with the liquidity factory, the Airlock facilitates the call to the liquidity factory and seeds the chosen liquidity bootstrapping pool (LBP) with a user-defined share of the token supply. The Airlock can poke the LBP to return the quote asset to then initiate the migration step. The LBP can additionally poke the Airlock to power the turnkey. It is possible for the liquidity generation step to fail if not enough proceeds are not generated, which is a parameter defined by the user. In this case, the Airlock will not migrate the liquidity.

To showcase the flexibility of the liquidity factory, a reference implementation of a `Uniswap v3` liquidity hook for Doppler is also shown in the project's GitHub.

4.1.3 Migration Factory Once required proceeds are generated through the LBP as part of the liquidity factory, the Airlock pulls the quote proceeds and remaining asset tokens to the migration factory. The migration factory facilitates the creation of the generalized AMM position while minimizing MEV lost. The current implementation targets Uniswap v2 as the destination protocol, leveraging its proven stability and broad ecosystem support. However, the modular architecture enables future expansion to support migrations to Uniswap v3, v4, or other AMM protocols as market needs evolve.

4.1.4 Timelock Factory The timelock factory creates a trusted contract for the token's ecosystem, which is used to mitigate short-term decision making for user safety. One big loss of value for a token's ecosystem is the burning of the LP shares done by most Protocols. This causes significant loss for the Protocol and turns a potential revenue generating asset into a drain. By creating a trusted contract that is locked by code, users can trust the liquidity for their project will not be removed, but token holders can mitigate significant losses to their ecosystem.

4.1.5 Vesting Modules Additionally, the portion of the tokens that are typically reserved for the developers of the token project are not given until the token project is fully liquid. Significant losses to retail users have come from sells in the bonding curve phase. A developer aiming for longer term support should be able to wait for the token project leaving the bonding curve. Plus, once a token project leaves the bonding curve, the price impact of a developer selling their shares is not as impactful due to the liquidity not being concentrated. We have also seen many token projects recover from the original developer selling their shares when the token project is liquid.

5 Protocol and Interface Fee Structure

Doppler implements a balanced fee structure designed to align incentives between protocol stakeholders while maintaining competitive market dynamics. The system incorporates both protocol-level and interface-level fees, with built-in mechanisms to ensure fair value distribution across the ecosystem.

5.1 Core Fee Architecture

The protocol establishes a maximum combined fee ceiling of 250 basis points (bps) to maintain market efficiency. Within this framework, Doppler's base protocol fee follows a dynamic structure: the protocol receives either 10 bps or 10% of the interface fee, whichever generates higher revenue. This design creates a sustainable funding mechanism for ongoing protocol development while preserving competitive market dynamics.

5.1.1 Interface Fee Implementation The fee structure allows interface providers to capture up to 225 bps in fees, creating substantial incentives for interface development and ecosystem growth. This generous interface fee allocation serves a strategic purpose: it encourages interface consolidation around standardized implementations, which enhances both trading safety and asset discoverability across the ecosystem.

By establishing significant interface fee potential, the protocol motivates developers to focus on interface optimization rather than competing through divergent market structures. This consolidation effect benefits the broader ecosystem by reducing market fragmentation and improving the overall trading experience.

5.1.2 Migration and Liquidity Fee Distribution The migration factory implements an additional 5% fee on swap activity, with the remaining fee revenue directed to the timelock contract. This fee split ensures sustainable funding for protocol maintenance while preserving significant value for long-term liquidity providers.

5.2 Design Vision

The fee structure's design produces several key benefits for the ecosystem.

First, the generous interface fee allowance encourages the development of high-quality interfaces, improving the overall user experience. By enabling meaningful revenue capture for interface providers, the protocol promotes investment in superior trading tools and safety mechanisms.

Second, the fee structure's emphasis on interface creation helps establish consistent standards across the ecosystem. This standardization simplifies the trading experience for users while reducing the technical overhead required to interact with multiple disparate systems.

The combination of protocol and interface fees creates a sustainable economic model that supports continued protocol development while maintaining competitive market dynamics. This balanced approach ensures the long-term viability of the ecosystem while preserving sufficient incentives for all participants."

6 Summary

Doppler represents a significant advancement in decentralized market infrastructure, introducing a comprehensive solution for the persistent challenges of liquidity bootstrapping and price discovery. The protocol's core innovation lies in its hybrid Dutch auction and dynamic bonding curve mechanism. This combination creates an efficient price discovery process that protects against common exploitation vectors while enabling automated market formation. Doppler's modular architecture extends beyond price discovery to address the broader challenges of market formation. The system's factory modules enable standardized token deployment with verifiable bytecode, automated liquidity provision, and high-quality standards.

References

[1] 1inch. 2022. *Fusion*. Retrieved Jul 16, 2024 from https://docs.1inch.io/docs/fusion-swap/introduction/

[2] Hayden Adams, Moody Salem, Noah Zinsmeister, Sara Reynolds, Austin Adams, Will Pote, Mark Toda, Alice Henshaw, Emily Williams, and Dan Robinson. 2023. *Uniswap v4 Core [Draft]*. https://github.com/Uniswap/v4-core/blob/main/docs/whitepaper-v4.pdf

[3] Hayden Adams, Noah Zinsmeister, and Dan Robinson. 2020. *Uniswap v2 Core*. Retrieved Jun 12, 2023 from https://uniswap.org/whitepaper.pdf

[4] Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. 2021. *Uniswap v3 Core*. Retrieved Jun 12, 2023 from https://uniswap.org/whitepaper-v3.pdf

[5] Hayden Adams, Noah Zinsmeister, Mark Toda, Emily Williams, Xin Wan, Matteo Leibowitz, Will Pote, Allen Lin, Eric Zhong, Zhiyuan Yang, Riley Campbell, Alex Karys, and Dan Robinson. 2023. *Uniswapx*. Retrieved Jul 16, 2024 from https://uniswap.org/whitepaper-uniswapx.pdf

[6] Elijah Fox, Mallesh Pai, and Max Resnick. 2023. Censorship Resistance in On-Chain Auctions. arXiv:2301.13321 https://arxiv.org/abs/2301.13321

[7] Frankie, Dan Robinson, Dave White, and andy8052. 2022. *Gradual Dutch Auctions*. Retrieved Jul 16, 2024 from https://www.paradigm.xyz/2022/04/gda

[8] Andrew Komo, Scott Duke Kominers, and Tim Roughgarden. 2024. Shill-Proof Auctions. arXiv:2404.00475 [econ.TH] https://arxiv.org/abs/2404.00475

[9] Ciamac C Moallemi and Dan Robinson. 2024. Loss-Versus-Fair: Efficiency of Dutch Auctions on Blockchains. *arXiv preprint arXiv:2406.00113* (2024).

[10] Sebeom Oh, Samuel Rosen, and Anthony Lee Zhang. 2023. *Digital Veblen Goods*. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4042901

7 Disclaimers

No Legal, Financial, or Investment Advice. This document does not constitute legal advice, financial advice, investment advice, trading advice, or a recommendation of any kind by Doppler, its affiliates, or any of their respective officers, directors, managers, employees, agents, advisors, or consultants. No information contained herein should be relied upon as the basis for any investment decision, contract, or any other decision regarding engagement with Doppler or any of its projects.

Forward-Looking Statements. This document may contain forward-looking statements based on assumptions and beliefs of Doppler, its officers, or its affiliates. These statements are subject to known and unknown risks, uncertainties, and other factors, many of which are outside Doppler's control, that may cause actual outcomes to differ materially from the projections or expectations set forth in such statements. Doppler makes no obligation to update or revise any forward-looking statements to reflect subsequent events, developments, or changes in circumstances after the date on which such statements were made, except as required by law.