Intuitive Launchpool Model

Joshua Ng, Melvin Zhang, and Meow

Studio Raccoons

Abstract

The Intuitive Launchpool Model is a new launchpool model that emphasises giving projects the flexibility to decide how they want to distribute liquidity over a price range. This is done using Meteora's DLMM Pools which allows projects to seed their tokens in a pool based on their strategy. Projects can put more liquidity at lower prices, allowing more traders to get their tokens for cheap, or put more tokens at higher prices, potentially securing more stable liquidity. This new model uses a price/bonding curve that describes how the price increases as more tokens are withdrawn from the pool. Projects can customise their price curve based on their strategy. The price curve uses 4 parameters, initial price, max price, curvature and tokens deposited. The parameters affect the price curve differently and will cause the end result to change. The price curve is then used to create a liquidity distribution graph which shows how the tokens are spread across the price range. This liquidity distribution graph is then used to create the DLMM Pool where the tokens are spread across the bins.

Overview

The Intuitive Launchpool Model presents a new intuitive model for launching tokens. Our model focuses on giving projects much more flexibility for token launches. One of the key mechanisms for the ILM launchpool is a single-sided Meteora DLMM pool, which allows projects to deploy liquidity based on their strategy. In the usual price/bonding curve models, like Uniswap, Balancer, and Curve, the models are highly rigid and inflexible, while Meteora's DLMM curve allows project teams to precisely distribute liquidity and tokens based on their preferences. The model allows projects to decide who they want to target, whether targeting retail investors or mitigating bot interference, this launchpool model puts the power back in the project's hands.

Each project and token is different and the launch model should not be the same for all tokens. The launch of a meme coin and $JUP should be very different as the tokenomics of meme coins and $JUP are different. Other launch models force inherently different coins to use the exact same model and don't allow the project owner to decide on key decisions. In the Intuitive Launch Model, projects can create a pool based on their valuation, the type of token, popularity and other factors that may come into play.

Since the setup of these pools is fully customisable, projects can choose exactly

how they want their curve to look and find out how much they can expect to raise. To assist teams in designing their price curves, we published a simulation site: https://ilm.jup.ag that allows you to play around with various inputs and understand how they differ. We want projects to design the price curve according to a set of highly intuitive parameters like initial price to start, max price to sell, and how fast the price should increase.

Why DLMMs

A Dynamic Liquidity Market Maker (DLMM) allows users to trade a pair of tokens just like an AMM. However, it works very differently from an AMM. While AMMs like Uniswap use a xy=k curve which uses a rigid and inflexible model which may not be suitable for every token launch, DLMMs allow projects to deploy liquidity at different prices and allow them to spread it out however they want, even if it does not follow our price curve model. This allows projects to customise their token launch based on different factors. For example, when there is a lot of demand, projects can put more tokens at higher prices, securing more stable liquidity. Traders also suffer from less slippage on DLMMs compared to AMMs and liquidity on DLMMs is more concentrated, allowing more effective use of liquidity.

The Price Curve Model

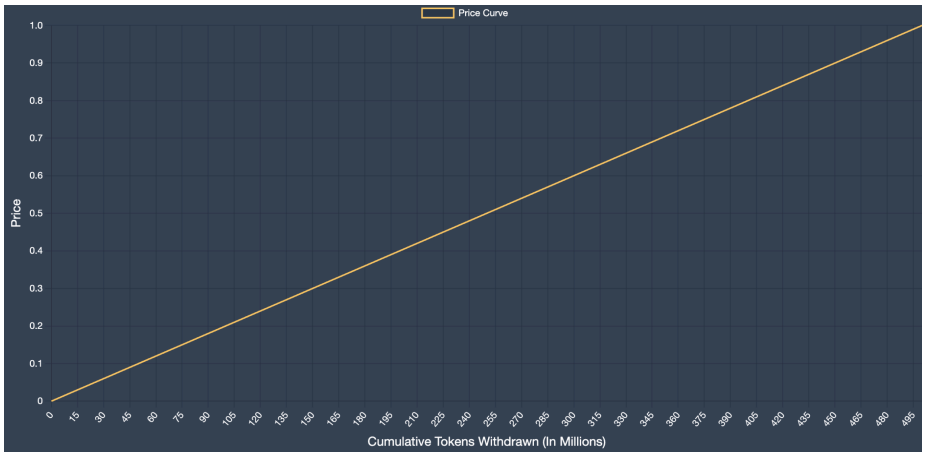
Unlike other models, we do not start with complex liquidity models which are hard to intuit. Instead, we start with a highly intuitive model to determine the price curve, based on the net amount of tokens in the pool vs USDC. The price curve is then used to create a liquidity distribution graph.

Our model uses bonding curves, which are mathematical curves that define the relationship between the price and the supply of a token within a particular system. The price of each token increases or decreases according to the price curve formula as tokens are withdrawn or deposited. Although the concept of bonding curves in DeFi isn't new and has been used by projects like Bancor, our model provides more flexibility and takes advantage of Meteora's DLMMs.

The price curve describes how the price changes as tokens are withdrawn from the DLMM pool, while the liquidity distribution graph describes how many tokens are available at each price point.

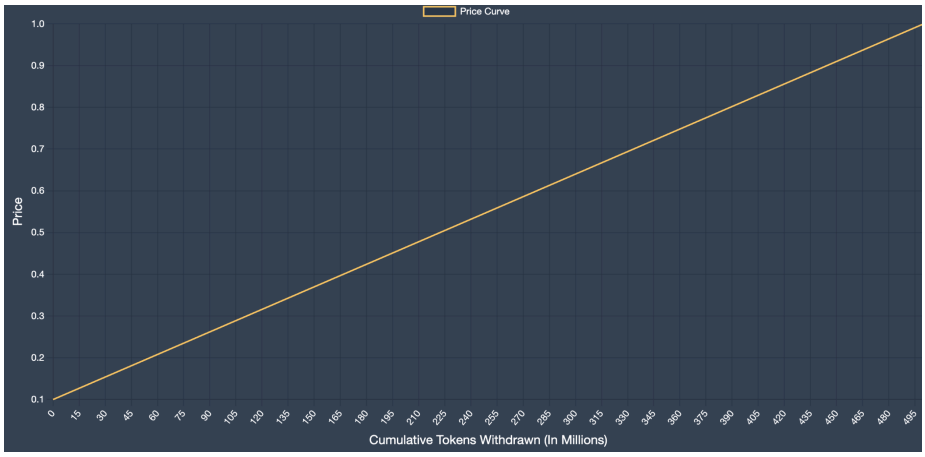
The price curve is determined by the formula $P = (M - I)\left(\frac{C}{A}\right)^K + I$, where: P is the price C is cumulative tokens withdrawn from the pool

The full formula can be complex, so we can first simplify it into a straight linear graph where the price is directly proportional to tokens withdrawn $P = \frac{MC}{A}$, where P is the price, M is the maximum price of the token in the pool, C is the cumulative tokens withdrawn from the pool and A is a constant that represents the total number of tokens in the pool.

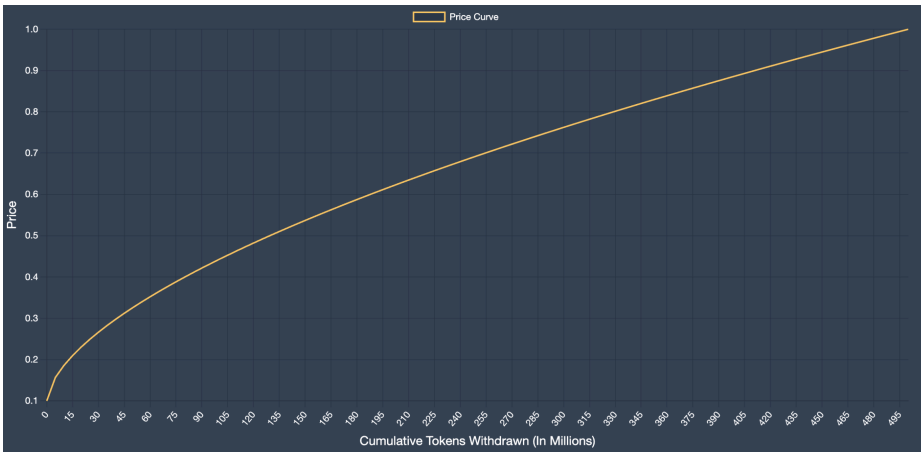


Here, the price increases at the same rate throughout the entire graph and the price starts from 0 USDC.

We can then introduce a new variable, I, the initial price. The formula thus changes to $P = \frac{(M-I)C}{A}$. With I, the Y-intercept increases but the max price doesn't change.



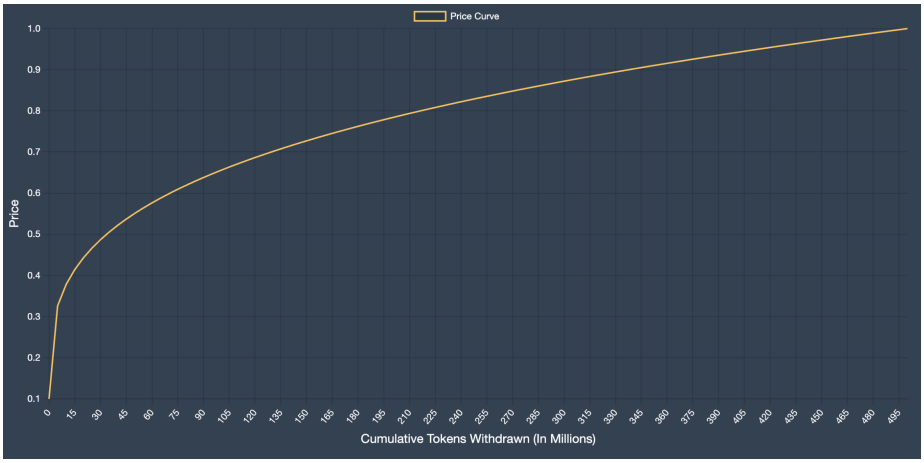
All our curves so far are linear, and so if we want a non-linear curve, we must introduce a new variable K, the curvature to create the final equation of $P = (M - I)\left(\frac{C}{A}\right)^K + I$. K is the exponent and allows the graph to be curved. this allows projects to focus liquidity at either higher or lower prices. The curve will either become concave or convex depending on the K value.



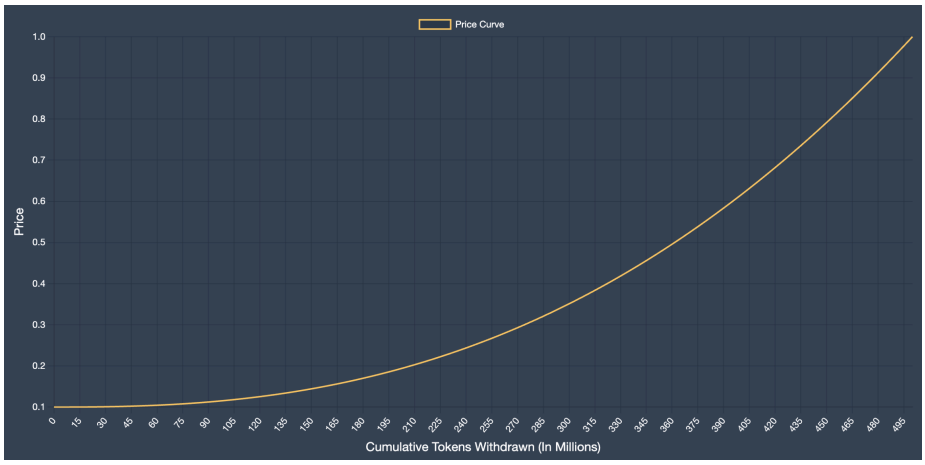
Here K is set at 0.6 and it causes the gradient to change but the initial and maximum price remain the same

The price curve can be adjusted using the following parameters which lends itself well to different strategies: K is the curvature which determines the steepness/gradient of the price curve I is the initial price the token will be offered at M is the maximum price the token will be offered at A is the total number of tokens in the pool.

For example, the following curve shows how the price will increase as more and more tokens get withdrawn from this pool.



This is a price curve with a steep early curve, where the price increases fast at the start. The initial price is set to 0.1 USDC and the final price is set to 1 USDC for this graph. The first token to be withdrawn is priced at 0.1 USDC. As the curve is steep at the start, by the time 10% of tokens are withdrawn, the price will move to 0.55 USDC. This graph gives less advantage to early buyers/bots as they can only buy a small amount before the price increases sharply. Thus they cannot get much at a very low price. This gives bots less advantage as there is not much low-priced liquidity for bots to snipe.

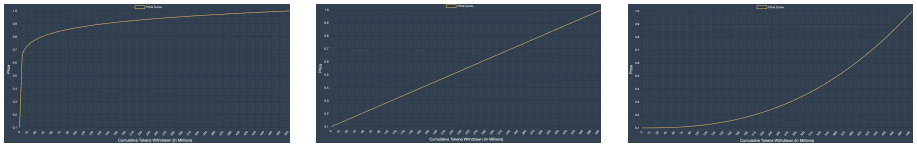


This is a price curve with a flatter early curve, where the initial price is 0.1 USDC And the final price is 1 USDC. The first token to be withdrawn is still priced at 0.1 USDC. As the curve is less steep at the start, if 10% of tokens are withdrawn, the price will only move to 0.12 USDC. This gives more advantage to early buyers as they can pick up much more tokens at cheaper prices. Bots can get many tokens for cheap.

Curvature Explained

Curvature decides the change in the gradient of the graph. High curvature means a larger exponent which leads to a concave graph. This leads to a graph where the price increases faster at the start and slower at the end. A low K value leads to a convex-shaped graph where the price increases slower at the start and faster at the end.

We have 3 examples here, all with different K values (all other parameters remained the same).



(a) K=0.1 (b) K=1.0 (c) K=2.5

The K value can greatly impact the speed at which tokens are withdrawn and the stable liquidity secured. This will be explained later.

Calculating Stable Liquidity Secured

One of the main reasons why projects provide liquidity using the price curve is to secure stable liquidity like USDC. The stable liquidity is secured as users withdraw tokens and deposit stable tokens into the DLMM Pool. The stable liquidity secured can be calculated using the integral of the price curve $\int_0^w (M - I)\left(\frac{C}{A}\right)^K + I\, dc$, where w is the cumulative tokens withdrawn

at that time. Projects can use this to find out how much they can secure given a certain valuation by first finding the cumulative tokens withdrawn(w) given a certain price. Then the number of tokens withdrawn can be used in the formula to find stable liquidity secure. This is an example of a price table which shows how many tokens are withdrawn and USDC in the pool at different prices. The USDC in the pool is calculated with the integral:

Price	Tokens Withdrawn (In Millions)	USDC in the Pool (In Millions)
0.2	0	0
0.25	1.5811	0.3727
0.3	8.9443	2.4277
0.35	24.648	7.5703
0.4	50.596	17.347
0.45	88.388	33.461
0.5	139.43	57.763
0.55	204.98	92.242
0.6	286.22	139.02
0.65	384.22	200.34
0.7	500	278.57

From the formula above, it can be deduced that an increase in I, M and/or A will increase stable liquidity secured while an increase in K will decrease stable liquidity secured.

Price and Token Table

The Price and Tokens Table are both tables giving essentially the same information but in a different format. The price table shows the stable liquidity in a pool and tokens withdrawn given a certain price. The Tokens Table shows the price and stable liquidity secured given that a certain amount of tokens are withdrawn from the pool. The Price Table highlights the trade-offs made when high or low K values are used. Here we have 2 examples, both with I=0.2, M=0.6 and A=200 but with different K values.

Price	Tokens Withdrawn (In Millions)	USDC in the Pool (In Millions)
0.2	0	0
0.24	0.092832	0.021423
0.28	0.93569	0.24472
0.32	3.6149	1.0567
0.36	9.4311	3.047
0.4	19.843	7.0212
0.44	36.436	14.014
0.48	60.91	25.301
0.52	95.06	42.411
0.56	140.77	67.136
0.6	200	101.54

Figure 2: K=0.3

Price	Tokens Withdrawn (In Millions)	USDC in the Pool (In Millions)
0.2	0	0
0.24	79.621	16.834
0.28	105.06	23.414
0.32	123.56	28.948
0.36	138.63	34.063
0.4	151.57	38.976
0.44	163.04	43.788
0.48	173.41	48.554
0.52	182.92	53.309
0.56	191.75	58.072
0.6	200	62.857

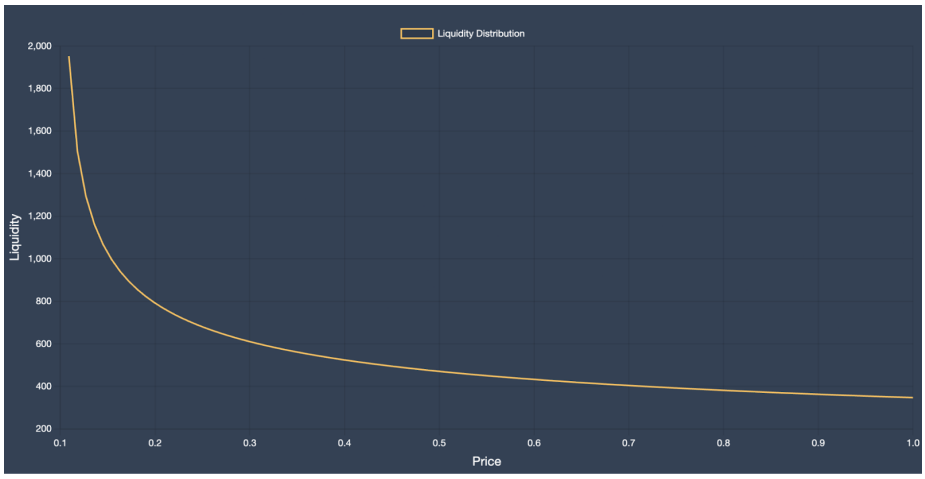
Figure 3: K=2.5

The table with K=0.3 shows that more stable liquidity is secured assuming all tokens are withdrawn. This means projects can potentially secure more funds with higher K values. However, when the price is much lower than the max price, and people stop withdrawing tokens after that price, the curve with K=0.3 will raise less as many tokens have not been withdrawn yet.

Liquidity Distribution

As mentioned before, the liquidity distribution describes the number of tokens available at each price point.

We get the liquidity distribution graph by first rewriting the equation $P = (M - I)\left(\frac{C}{A}\right)^K + I$ to express C in terms of P which is $C = A\sqrt[K]{\frac{P-I}{M-I}}$. The derivative of that is then the liquidity distribution which is $\frac{A\sqrt[K]{\frac{M-I}{P-I}}}{K(P-I)}$. This represents the amount of tokens available at each price. The integral of the graph is the total number of tokens in the pool initially and the number of tokens available within a price range can be found with it. This will help us generate the DLMM Bins later on. The liquidity distribution graph for a graph with a larger K value like the one shown below, is higher at the start and lower at the end. This means that more tokens are available at low prices and less at higher prices.



Liquidity Distribution Graph: X-Axis: Price, Y-Axis: Liquidity / Tokens Available

Translating Liquidity Distribution into DLMM Bins

Introduction to DLMMs

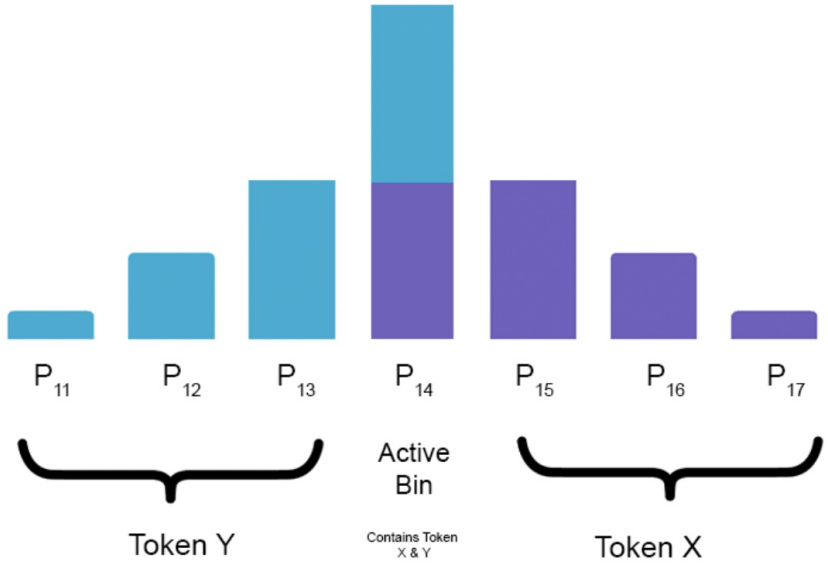
Meteora's Dynamic Liquidity Market Maker (DLMM) involves dividing the total liquidity range of an asset pair into smaller segments called "bins". Each bin has its own unique exchange rate and ID. This segmentation allows for precision in how and where your tokens are offered, adapting dynamically as the market moves.

Each bin can contain either token but most bins only contain 1, with the bin that has the current exchange rate having both tokens. This bin is referred to as the active bin.

Although users can technically trade in any bin they want to, most users just swap tokens in the active bin as it is the bin with the best rate while having the token they want. When tokens are traded in the active bin, the ratio of tokens in the active bin changes. When all of either tokens in the active bin are withdrawn, an adjacent bin becomes the new active bin and the exchange rate changes.

The price difference between adjacent bins is defined by the bin step, which is set by the pool creator.

This is a simple example of a DLMM pool where there are bins with tokens X and Y.



Here, there are 7 bins, each with an exchange rate represented by P and the active bin is the bin that has both tokens and the bin where the most activity occurs. The active bin changes based on the current price and when all of either token in the bin is withdrawn, the active bin moves. For instance, when someone withdraws all the token X in the active bin, the active bin moves to the bin to the right and the price of token X relative to token Y increases. The former active bin now only contains token Y and the new active bin will contain both tokens.

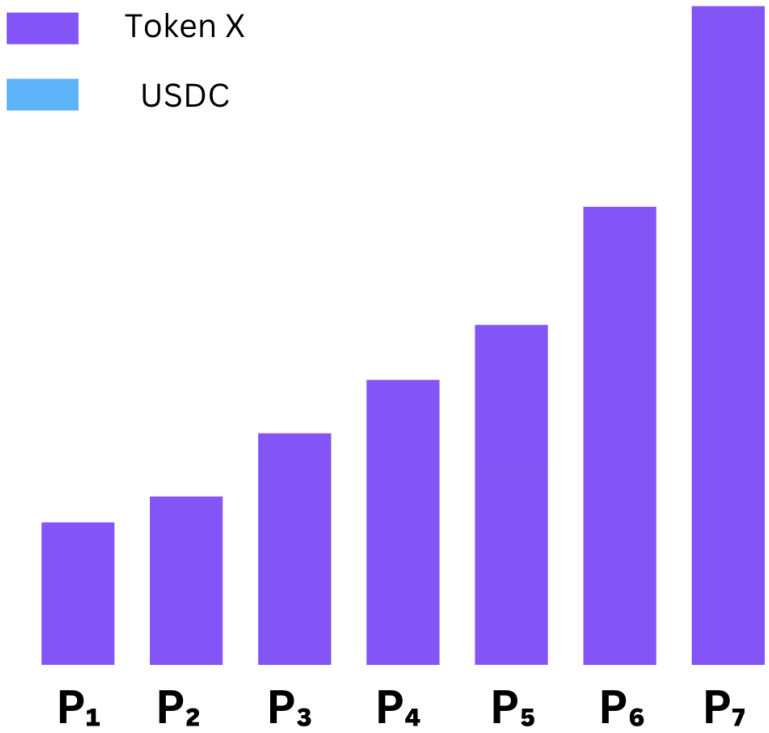
The size of the bins represents the liquidity in the bin. The sizes are calculated in terms of the quote token which is set by the owner. Let's assume token Y is the quote token. The bin size is then calculated using $T_X P_X + T_Y$, where T_X is the number Of Token X in the bin, P_X is the exchange rate of X to Y in that bin, T_Y is the number of Token Y in the bin

Creating a single-sided DLMM Pool with Token X and USDC

Here we are creating a launchpool for token X using our price curve model. The DLMM pool is a little different from the one shown above as it starts off single-sided. That means that the pool will initially only have token X and will have no USDC. When the token goes live, traders are allowed to trade and withdraw token X and deposit USDC, the active bin shifts to reflect the current price.

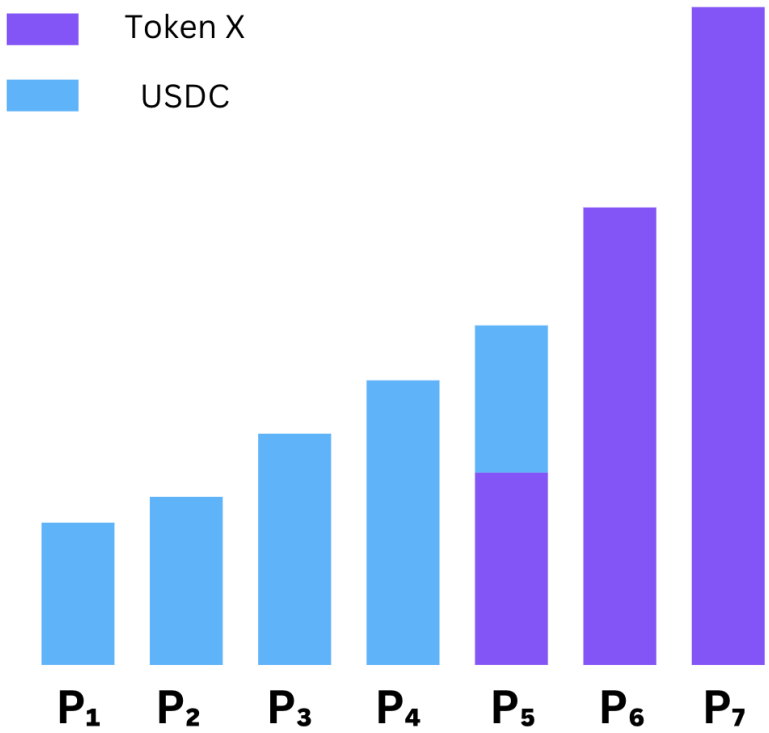
Using the price curve generated, the tokens can be distributed into a single-sided liquidity position on the DLMM curve. Each bin represents a small portion of the liquidity distribution graph and will have the number of tokens of the range it represents. Therefore the integral of the liquidity distribution is used to find the amount of tokens in that bin. The formula to find the tokens in a bin is, $\int_l^u \frac{A\sqrt[K]{\frac{M-I}{P-I}}}{K(P-I)} dP$, where u is the price of the next bin and l is the price of the bin.

Therefore we find the number of tokens available within that price range, which is represented by the bin. The bin size is generated using the formula bin price x number of tokens in the bin.



Initially, the pool will only have token X so it will look like this, where there is only X in the pool and there is no USDC.

When users start withdrawing X and depositing USDC, the active bin moves to the right, and the active bin will move alongside the price.



This is what the DLMM pool looks like when users have bought all the X in the first 4 bins and have bought some of the X in the 5th bin. Thus the current active bin is the 5th bin at price P_5 and the current price is P_5.

In the actual one-sided DLMM curve, there will be hundreds or even thousands of bins.

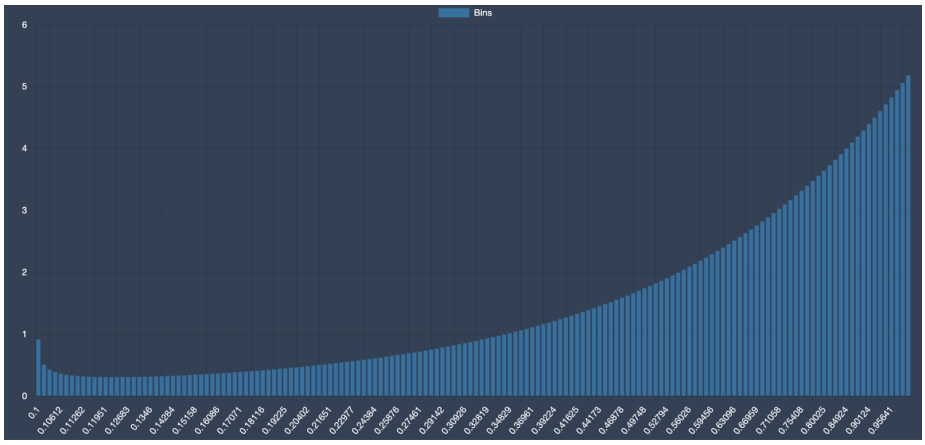
In the DLMM pool, the price increment from 1 bin to its neighbour is not constant, in fact, the price difference increases exponentially. This is because when prices are higher, we don't care about small increases as much. For example, if SOL is 20 USDC, a price increase of 5 USDC is of large importance as the price increases by 25%. However, if the price of SOL is 200 USDC, a 5 USDC increase is not as significant as the price only increases by 2.5%. Thus, our bins price increases exponentially to accommodate for the granularity needed. The way we calculate the increment will be explained later.

Each bin is identified by a unique integer ID, and the bin to its right possesses an ID that is one more than its own. DLMM Pool Bin ID can be converted into the bin price using the formula $Price = \left(1 + \frac{step}{10000}\right)^{ID}$ and ID can be obtained from price by using the formula $\text{ID} = \left\lfloor \frac{\log(Price)}{\log(1+\frac{step}{10000})} \right\rfloor$.

We create the DLMM pool by first getting the ID of the first bin. This is done by plugging the initial price into this formula, $\text{ID} = \left\lfloor \frac{\log(Price)}{\log(1+\frac{step}{10000})} \right\rfloor$. We then find the ID of the last bin by plugging in the max price into the same formula. We then find all the integers in between the first ID and last ID to

generate the IDs for all the other bins. Using these IDs, we use the formula $Price = \left(1 + \frac{step}{10000}\right)^{ID}$ to find the price for all the bins.

For example, if the price range is from 0.1 to 1 and the bin step is 150, the first bin ID is -154. The last bin ID is 0. Thus, in this case, there will be 155 bins in the pool with IDs of -154, -153, -152....0. This is what the bins look like with a K value of 1.6 (Note that there are many bins which makes it hard to see each individual bin)



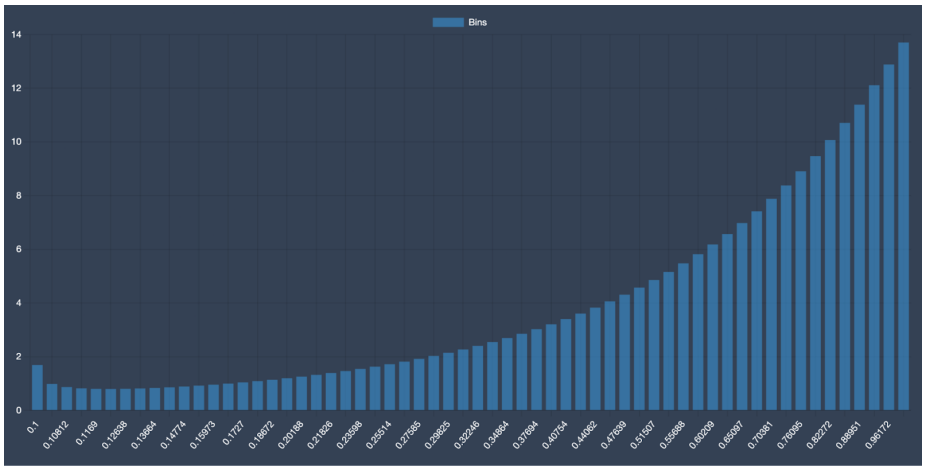
Each bar below represents a bin and buyers start buying from the first bin. The vertical axis is liquidity in USDC and the Y axis is bin ID.

The DLMM Bin distribution also does not need to follow any of our proposed price curves and liquidity distribution graphs. Liquidity can be distributed however the project wants to and does not have to be limited to any curve formula.

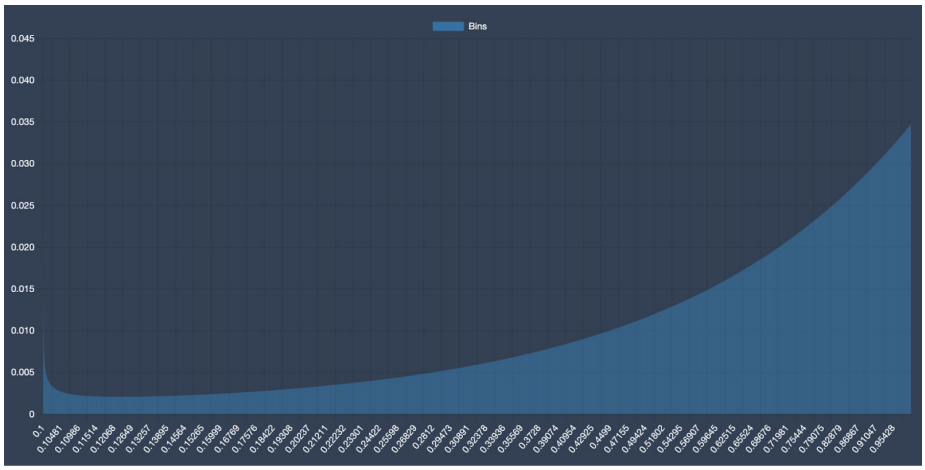
Due to how the bin ID is converted into bin price and vice versa, the bin step determines how many bins there are.

Thus, a larger bin step means that the difference between adjacent bins are generally larger

This is what the bins of a pool with bin step 400 look like:



This is what the bins of a pool with bin step 1 look like:



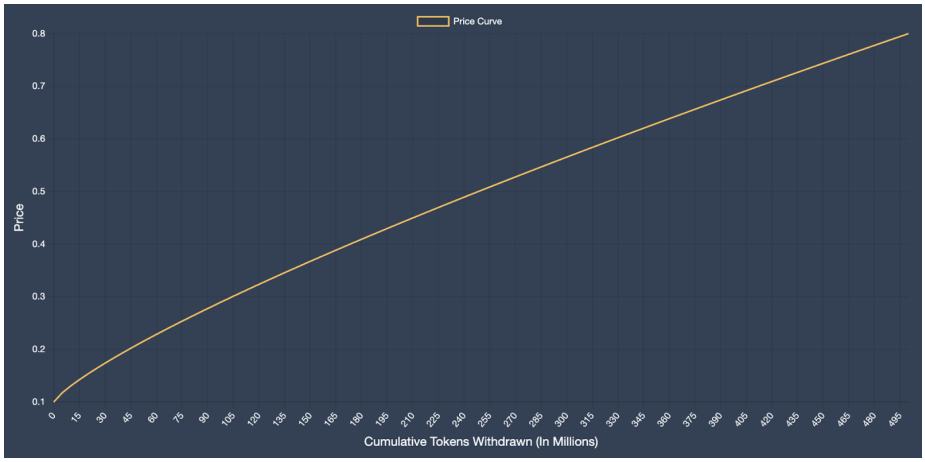
Smaller bin steps tend to follow the price curve more accurately, however, the discrepancy between the price curve and DLMM is very small as long bin step is not extremely large. Small bin steps are mostly for pools that require very granular increases from one bin to another, like USDC-USDT Pools, where extremely small price changes have large impacts. However, for token launches, a small bin step has consequences. A bin step that is too small may cause a higher rate of transaction failure as the active bin changes too fast, the active bin might have changed before the transaction finalises, causing it to fail. This is especially the case for projects with high demand, which causes the active bin to move extremely fast. Smaller bin steps also prevent liquidity providers from putting liquidity over a wide range and only allow them to put liquidity across a small range.

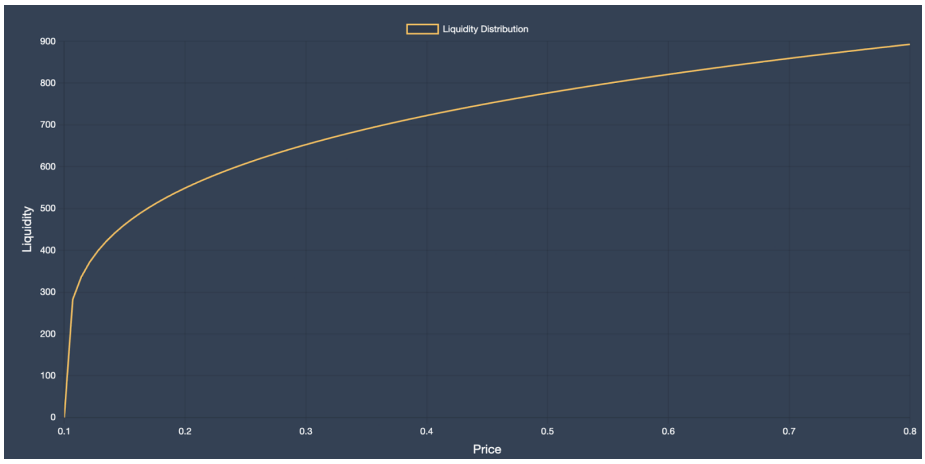
Example using $JUP

This is a curve that has been proposed for $JUP and was used for mockJUP: Note that these may not be the values used for $JUP This curve gives less advantage to early buyers and bots as the price increases faster at the start and

more liquidity is towards the end

(K=0.8, I=0.1 M=0.8, A=500)





The price curve shows how the price increases as more tokens are withdrawn from the pool. The amount it increases by is affected by K, which affects the gradient of the curve.

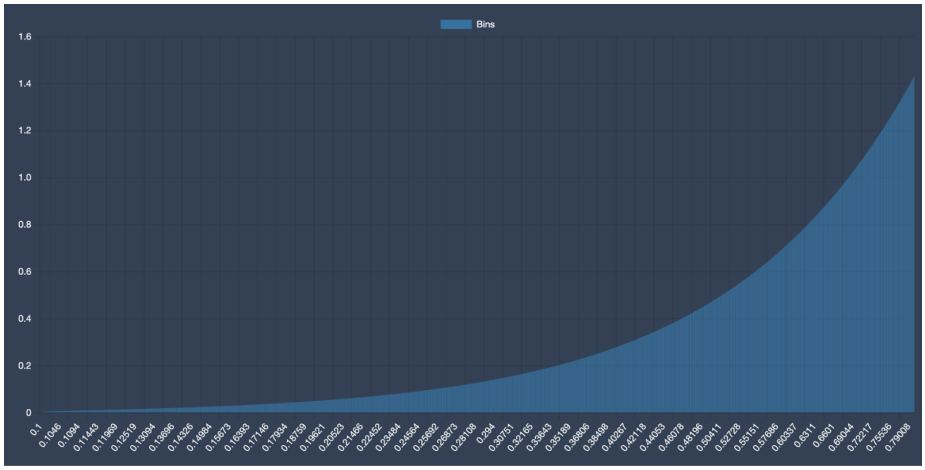
The liquidity distribution graph shows the number of tokens that can be withdrawn or liquidity at different prices.

In the example above, there is more liquidity at higher prices compared to lower prices as liquidity increases with price.

At the end of the curve, when all tokens have been purchased (and no additional tokens added to the pool), the price will be at the maximum price (M). Using the curve above, when 0 JUP tokens have been withdrawn, the price is at \$0.1(I). When 250 Million JUP tokens have been withdrawn, the price rises to around \$0.5 and at the end when all 500 Million JUP tokens have been withdrawn, the price is at \$0.8 (M).

Using the parameters above, we can generate a DLMM Pool with a bin step of 25 that looks like this:

The total number of bins inside is 833.



Although in this chart it seems that there is way more liquidity at higher prices as the bins to the right are much larger, this is because the bin size is calculated using bin price times the number of tokens in the bin. Thus the bins to the right are larger as they have a larger price. Bins to the right also have more tokens as they cover a larger price range and represent a larger portion of the liquidity distribution graph

With a fixed range, bin steps will affect the number of bins. If the bin step is larger, there will be fewer bins, and vice versa.

Summary

The ILM launchpool Model focuses on providing liquidity and bootstrapping rather than price discovery. It allows projects to customise their price curve to best suit their needs instead of using other launch models which are rigid. We design the price curve using parameters that will change the shape of the curve and thus lead to different outcomes. The price curve can be then used to create a liquidity distribution graph which then is used to create a single-sided DLMM pool where traders can withdraw tokens from. The curve can be designed to also benefit different groups differently and can be designed to either give more or less advantage to early buyers.