# Autonolas Tokenomics v1

## - Towards a Self-Sustainable Autonomous Services Ecosystem via Software Composability Incentives -

Mariapia Moscatiello, David Minarsch, David Galindo

Valory AG, Zug, Switzerland
info@valory.xyz
https://valory.xyz

**Abstract.** This yellow paper offers a comprehensive overview of the token-economics model of the Autonolas protocol. The protocol aims to incentivize software composability to foster the growth of a self-sustainable ecosystem of autonomous applications.

**Key words:** token-economics, software composability, autonomous applications, protocol-owned liquidity, protocol-owned services, open-source code.

## 1 Context

Today's world is increasingly reliant on software. Indeed, humans have strived for centuries to automate repetitive processes for their economic benefit or saving time, software being its epitome. Despite that, more often than not, humans are *paradoxically* placed at the service of machines. It is our conviction that the focus is shifting from automation towards higher levels of machine autonomy at the service of human *autonomy*.

We believe the latter can be accelerated by creating *autonomous applications* or *services* that, by design, require little to no input from humans, run continuously, and take their own actions to benefit humans. Autonomous services are, additionally, capable of complex processing and have composability properties to facilitate growing their number with less and less effort. They can be operated by one or different entities transparently, and can achieve varying levels of decentralization. When operated by groups with open participation, the reliable and collaborative aspects of autonomous services offer great potential for the benefit of society. A very relevant opportunity is that of co-owned and co-operated AI as autonomous services (see in [24, 25] for more details).

Autonomous services can be implemented as *agent services* [8]. As such, they take the form of a multi-agent-system [4] that can be anchored to a settlement layer [11] but are primarily run off-chain, e.g. outside the settlement layer environment (such as a public blockchain). The execution of an agent service logic can be coordinated between participating *agents*, and the service intermediate states are replicated on a *short-lived* blockchain (also called *consensus gadget* [8, 5]), specific to each service. The most important intermediate values obtained in a given service can be secured on a public (or private) blockchain. The owner of the service, e.g. the entity that provides service specifications, deploys it and manages it, among other decisions and management activities, chooses in advance which values will be regularly committed to the blockchain.

For more information on the technical architecture of autonomous services see Chapter 2 in [23].

The foundation of a self-sustainable ecosystem of autonomous services is at the heart of the Autonolas protocol, and consists of the following elements:

- An open-source software stack to realize autonomous services form composable parts
- Software components[1] that provide specific functionalities and can be composed into software *agents*[2]. Agents can be themselves composed into distributed software services, e.g. autonomous services
- A protocol, that is blockchain-agnostic and could eventually be deployed on all major smart-contract blockchains, that provides the tools for the coordination, security, and management of autonomous services

[1] Software components are throughout the paper referred to as *components* or *agent components*.

[2] Agents are also called *canonical agents*.

– A tokenomics aiming to incentivise software composability to grow a sustainable autonomous services ecosystem

## 2 Introduction

The token-economics model that we present in this section lies at the heart of the Autonolas protocol[3] and has the aim of enabling a self-sustainable ecosystem of autonomous services[4].

The key resource in increasing the number of software services and/or code components in Autonolas is that of software developers. As such, the model gives due consideration to attracting and retaining software developers and facilitating the composability of their code contributions, which enables software code to become more than the sum of its parts. The model is additionally geared towards attracting capital in order to make this capital useful, by pairing it with useful code. Finally, the model introduces a novel mode of production for the ecosystem of autonomous services to become sustainable. We expand on these objectives next.

### 2.1 Model objectives

**• Pairing Capital and Code.**

The first objective of the tokenomics model is to enable the pairing of capital and code in a permissionless manner. This pairing is essential because code on its own does not communicate its value, and capital on its own is not productive. Combined, these tenets reinforce each other. Indeed, by pairing quality code with capital, we partially tackle a key problem faced in open-source development: developer time is a scarce resource subject to insufficient remuneration. A bonding mechanism can be used to grow the protocol's capital in the form of protocol-owned liquidity (and this bonding mechanism is, as we shall see, designed in such a way that allows bonded capital not to exceed code usefulness). And a novel "staking" model for code allow developers to track their code contributions on-chain (e.g. on Ethereum) and receive rewards for its usefulness[5].

**• Enabling Protocol-owned Services (PoSe).**

The second objective is to create a flywheel that attracts increasingly more value and provides truly-decentralized autonomous services, owned by a DAO[6], operated by ecosystem actors, and coded by the ecosystem developers. The novel

[3] The Autonolas (on-chain) protocol was deployed on the Ethereum mainnet in the Summer of 2022.

[4] A high level summary of this tokenomics is also provided in the Autonolas whitepaper, see Chaper 3 in [23].

[5] Code usefulness is measured in terms of the relative contribution of the code to the donations accrued by the Autonolas protocol. The term usefulness and its related measures will be more clear in what follows, cf. Sections 3.1 and 5.1.

[6] Decentralized Autonomous Organization.

tokenomics primitive, defined as protocol-owned service (PoSe), enables the DAO that manages the PoSe to own a productive service and derive profits from it.

The flywheel can be created by acquiring and productively deploying in the ecosystem the profits accruing from Autonolas' PoSes and the donations arising from autonomous service (see Section 2.3).

• **Incentivising Composability.**
The third objective is to enable and incentivize software composability. In a nutshell, "a platform is composable if its existing resources can be used as building blocks and programmed into higher order applications." Composability of software components in an ecosystem enables exponential returns. This composability is enabled by representing code and services as NFTs registered on a public blockchain and be combined together into higher-order applications. The composability is then extended by the tokenomics model that allows developers to track their code contributions on-chain and receive rewards for their usefulness (cf. footnote 5 above).

To accomplish these objectives, a token is used as a coordination mechanism. This explains why we talk about token-economics model or, as more widely known, tokenomics model. Notably, Autonolas tokenomics coordinates the above objectives via the OLAS token introduced below.

### 2.2 OLAS token

OLAS is a tradable utility token that provides access to the core functionalities of the Autonolas protocol. The token follows the ERC20 standard [13] and has been deployed on the Ethereum mainnet (OLAS token). OLAS follows an inflationary model that accounts for the economic primitives enabled by this tokenomics, such as the bonding mechanism and specific OLAS top-ups, e.g. extra incentives that developers can eventually receive (cf. Sections 3.1 and 5.2).

However, the bonding mechanism and top-up incentives are designed to not lead to an unrestricted amount of OLAS being issued. Notably, an inflation schedule is used to limit the maximum of what can be minted every year (cf. Section 4).

Regarding incentives, 47.35% of the fixed token supply (i.e. 1 billion OLAS) of the first 10 years is programmed to be distributed to the ecosystem. After 10 years the maximum token inflation per annum is capped at 2% and the DAO governance can opt to further reduce it. The aim is to have an $s$-shaped curve of token emissions to allow for the ecosysistem to organically grow over time. The rest of the fixed token supply is distributed as follows:

- DAO Treasury: 10%
- Development reserve: 10%
- DAO founding members: 32.65%

Fig. 1 gives a graphical representation of the OLAS allocation. Note that OLAS initially allocated for founding members are, in practical terms, locked, and unlock over the course of four years.

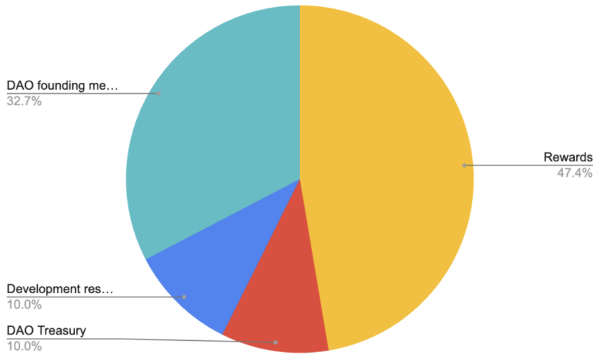


**Fig. 1.** OLAS token allocation, where Rewards are obtained by minting

### 2.3 Business Model

In order to build a sustainable autonomous services ecosystem, the majority of protocol profits are expected to arise from the following three primary contributors.

- Protocol-Owned-Services: The Autonolas DAO can run services whose profits are then returned to the protocol and are owned by the protocol treasury. Towards the growth of the ecosystem, the DAO can choose to share some of the profits - through the donations mechanism - with the developers who contributed useful code to the PoSes.
- Protocol-owned liquidity: This can either be deployed to third-party protocols for purposes like providing liquidity in decentralized exchanges or put to work in protocol-owned services. All profits returned from PoL are under the protocol's treasury ownership.
- Third-party services (incl. third-party PoSes): A limited amount of the protocol's profits likely comes from the treasury's fee on the voluntary donations from third-party services. Specifically, anyone can use the donation mechanism to provide a donation for services registered on the Autonolas protocol. The donation mechanism allow then to calculate a fair distributions of the donations to share among the developers (owner of staked code NTFs) who contributed with useful code to the useful services. The DAO can opt to turn on fees on the amount donated. Such fees are then under the protocol's treasury ownership.

Slashed agent operator bonds (cf. Section "Agent Operator Perspective" in [23]) are another, likely limited, source of treasury profits.

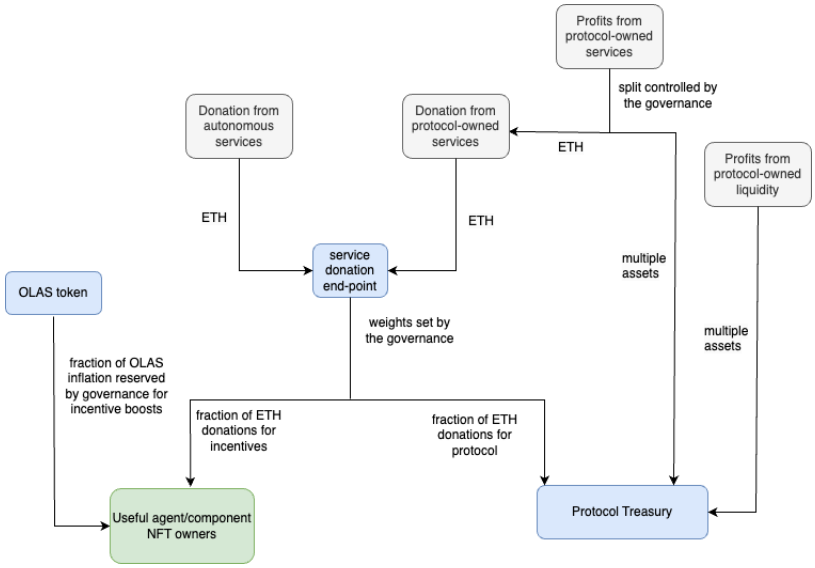


**Fig. 2.** Value flow: from the donator to useful code rewards and to the protocol treasury. During the initial 10-year growth phase the code rewards can be topped up with OLAS issuance.

## 3 Autonolas tokenomics in a nutshell

We recall that the focus of this tokenomics model is to increase the number and quality of autonomous software systems (or autonomous services). We believe that a crucial step in achieving this goal is to place software composability at the core of the design of autonomous services. In other words, software components of autonomous services can be composed into larger components or into agents, agents into services, and services can be easily composed with each other. The mathematical model leverages these properties of composability in order to incentivize software development and push further composability toward the exponential growth of functionalities. Moreover, the mathematical model and the Autonolas protocol are designed in such a way, the tokenomics objectives are reached as autonomously as possible.

To accomplish Autonolas' objectives, our proposed tokenomics model primarily leverages the following primitives:

i. **Staking model for agents and components code.** Developers of agents and components can register their code on-chain by using Autonolas registries (see [9] for the registries repository and "On-chain Protocol to Anchor Agent Services" section of [23] for more details on the technical architecture). By means of NFTs [14], the registries allow developers to uniquely represent on-chain their code that is stored off-chain. Then the developer can accrue incentives proportional to their code contribution.
ii. **Bonding mechanism.** The protocol can grow its own liquidity by incentivizing liquidity providers to sell their liquidity pairs (with one of the tokens in the pair being the protocol token, e.g. OLAS-ETH) to the protocol in exchange for OLAS at a discount.
iii. **Protocol-owned services (PoSes).** These are autonomous services, owned by a DAO, run by ecosystem operators, and implemented by (third party) developers. This novel primitive enables the DAO that created the PoSe to own productive autonomous services and derive donations from them.

### 3.1 How is the staking of the code incentivized?

The workflow for developers can be summarized as follows.

1. Developers build their components or agents as code.
2. Developers can have their code uniquely represented by NFTs on-chain. In a sense, developers are staking their code NFTs in favor of the ecosystem.
3. Developers (that may and may not correspond with any of the developers that created the underlying component) can use staked components and combine them together into new components or agents and then stake their new code NFTs[7]

[7] Technically, developers can also decide to not stake their code contribution, though tokenomics only incentivize staked code.

Service owners can therefore use staked agents and components to build autonomous services. Using the Autonolas protocol, service owners can register, manage and secure their own services on-chain.

Once the code can be uniquely tracked on-chain through their representation as an NFT, Autonolas tokenomics allows to automatically measure the code NFT's contribution to the ecosystem and, based on that, compute the corresponding code NFT incentives.

The business model described in Section 2.3 shows that donations accrued from autonomous services provide one of the primary contributions to the protocol. Each donation (currently accepted in ETH) sent from a service owner (or any other entity) is an indication of the *usefulness* of the service, and such donation is considered as a proxy for the ecosystem of the code making up the *useful* service. Therefore the *usefulness of code NFTs* is measured in terms of the donations that such code NFTs facilitate (see Section 5.1 for a full account of measures for useful contribution).

Autonolas tokenomics autonomously allocates a proportional share of the donation accrued by the protocol to useful code NFTs that facilitated it (see reward formulae in Section 5.2).

Moreover, whenever the agent or the component NFT is referenced in a useful service whose owner or whose donator locks OLAS for Autonolas governace token veOLAS[8], the Autonolas Protocol tops up the share of donations going to developers that own the agent or component NFT with freshly minted OLAS. The OLAS top-up acts like an "NFT staking reward": e.g. developers stake their "useful" code NFTs for autonomous services and receive a share of the OLAS inflation in exchange.

Incentives in ETH and top-ups in OLAS create a "push" and "pull" dynamic. The "push" results from the potential yields in ETH tokens that attracts developers not yet interested in the protocol token. Other than the potential yields in ETH and OLAS boosts, the "pull" also results from the potential governance impact on adjusting yields and reward boost for useful code NFTs that developers can accrue by locking their OLAS boosts.

### 3.2 How and when is the bonding mechanism incentivized?

Another source of the protocol's capital, as mentioned in Section 2.3, is provided by leveraging a bonding mechanism for protocol liquidity inflow. Similarly to what Olympus DAO bonding offers [15], bonds allow users to buy OLAS from the protocol at a discount by exchanging it for liquidity assets (LP tokens, e.g. ETH-OLAS). Autonolas bonds have a certain time to vest, called the vesting

[8] Note that there is a minimum threshold of veOLAS which should be owned by the service's owner in order to enable OLAS top-up.

period. After the vesting period expires, the bonder can claim the OLAS in full. The main difference with Olympus DAO is that the Autonolas bonding mechanism is built in such a way that bonded capital in the protocol does not exceed code usefulness. Moreover, as already mentioned in Section 2.2, Autonolas bonding mechanism not brings an unrestricted inflationary schedule with large amounts of new OLAS issuance in short time. Notably, the inflation schedule described in Section 4 is used to limit how much can be minted every year at maximum. In turn, since bonding implies minting new OLAS tokens, there is a maximum amount that can be bonded every year.

Specifically, a control mechanism is introduced to ensure that the growth of Protocol-owned Liquidity ("*Capital*") is proportional to the potential growth of the components and agents ("*Code*") productively deployed in services in the ecosystem. Such a control mechanism is enforced by several built-in on-chain metrics which are used to estimate the potential for the production of code by means of a production function $PF(.)$ (see Section 5.3 for more details). Currently, the mechanism can be described as follows: the larger the current value of $PF(.)$ is, the larger the discount at which bonders receive OLAS, and the smaller $PF(.)$, the smaller the discount, while respecting the inflation schedule.

This control mechanism enables "push" and "pull" dynamics when there is a high potential output in the production of useful code that, in turn, could be productively deployed in autonomous services. The "push" results from an advantageous discount that a bonder can receive in purchasing bonds. The "pull" results in the willingness of the OLAS holder to provide liquidity to the protocol thus contributing to token price stability.

### 3.3 Incentive compatibility

This tokenomics model is aimed at aligning the goals between the various players in the ecosystem. Such goal alignment can be enforced through economic incentives designed such that the overall higher expectation of benefit arises from conducting honest rather than adversarial actions. Players that are incentivized with rewards, or that are crucial for reward allocation, are code developers (e.g. agent and component NFT owners), autonomous service owners, and Autonolas DAO members.

**Developers perspective.** The more useful services reference a code NFT, the more incentives are allocated to that code. As more code is staked, more services can be created and, in turn, the higher is the likelihood that such services are useful and larger can be incentives for the useful code.

Moreover, the amount of gas needed for creating services increases with complex code, e.g., components and/or agents with a lot of dependencies. So the likelihood to have more references for staked code increases when the developer built an agent or a component by adding the minimal required information and the minimum number of dependencies to make such code self-consistent (cf.

Section D in the Appendix for further details). Therefore the developer benefits from correctly staking their code and adding due diligence into their development process.

**Service owner perspective.** Continuous staking of newly developed or well-maintained agents and components allows service owners to update their services and or deploy new ones. This in turn increases the likelihood that service owners have competitive services with higher chances of obtaining increased revenue from their users.

As more donations are accrued by the protocol from a service owner, the more incentives are allocated to the code referenced in such a service, and in turn more developers are incentivized to continue deploying and staking new code.

Thus, service owners not only have an interest in donating to the protocol in order to increase developer incentives, but they are also pushed to stake OLAS for veOLAS (giving them eligibility to participate in the Autonolas DAO governance) to adjust and boost incentives for owners of code NFTs.

**DAO members perspective.** Autonolas DAO is composed of veOLAS holders (cf. section "Level 2: Locking" in [23]). Holders of veOLAS are incentivized to maximize the utility of the OLAS token which they have locked away in return for governance rights. Notably, this can be done by wisely choosing governance parameters that determine how donations accrued by the Protocol and newly minted OLAS are shared across two recipients (Code owners of NFTs and protocol treasury) of rewards in Autonolas tokenomics.

Moreover, upon activation of off-chain signaling (cf. Section "Level 3: Off-chain Signaling" in [23]), veOLAS holders will[9] also be able to shape the protocol by signaling the positive/negative contributions of agents, components, or services which complements on-chain built-in code usefulness measures.

### 3.4 Ecosystem Flywheel

Autonolas provides a token-mediated coordination mechanism for the creation, operation, and sustainability of autonomous services. This is achieved via a flywheel designed to attract increasingly more value and provide more PoSes, owned by DAOs, operated by participants in the ecosystem, and implemented by developers.

At a high level, the Autonolas flywheel looks as follows:

1. A developer builds a component or agent code and stakes the associated code NFTs on-chain.
2. The component/agent enables more functionality which adds value to the ecosystem.
3. Autonolas DAO or any other DAO can operate its own services made up of agents and their components.

[9] This functionality is not active in the early days of the protocol's operation.

3. Autonolas protocol accrues donations from its own PoSes and/or third-party PoSes.
4. Autonolas protocol rewards useful component/agent NFTs owner with a share of accrued donations, and eventually, tops up such rewards with newly minted OLAS.
5. Bonders bring capital (in the form of LP tokens) to gain a discount on the OLAS price. The discount increases downstream protocol donations and/or encourages more useful code to be staked. Protocol-owned liquidity and OLAS price see upward pressure.
6. The deployment of Protocol-owned Liquidity and the increase of OLAS price attracts developers to contribute with more Code (which closes the loop of the flywheel).

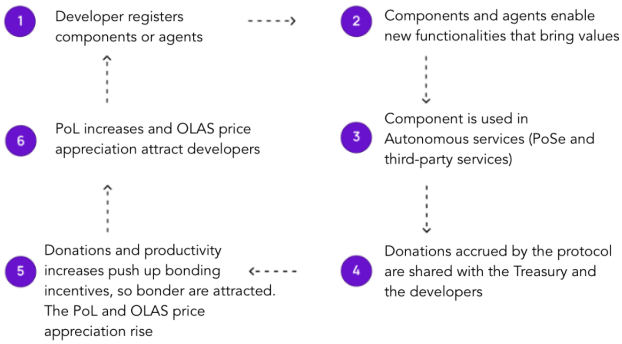


**Fig. 3.** Ecosystem flywheel

## 4 OLAS Inflation Schedule

As discussed earlier (cf. Section 2.2), OLAS is an inflationary token. During the first 10 years, 47.35% of the total token supply (1 billion) is expected to be issued via bonding (cf. Section 5.3) and component/agent top-ups (cf. Section 5.2). Further, the protocol is programmed to maintain a low ($0 - 2\%$ p.a.) level of token inflation thereafter. Issuing most of the OLAS before Autonolas has full recognition is to be avoided whenever possible. Hence during the first years from the protocol's launch, inflation is relatively small, with the highest inflation ratio taking place in years 2 to 4. Such a choice implies an $s$-shaped curve of token issuance.

Technically, the *inflation schedule* can be described by means of the following parameters:

- *First year inflation rate*: the percentage of the initial token supply that is minted during the year of the OLAS token launch
- *Second year inflation rate*: the percentage of the token supply that is minted during the second year
- *Initial inflation rate*: the percentage of the token supply that is minted for the third year
- *Dis-inflation rate*: the rate with which inflation is decreased every year in the range comprising year 3 to year 9
- *Long-term inflation rate*: tokens minted in the long run

**Fixed parameters** After having simulated various inflation schedules the following parameters have been proposed:

- *First year inflation rate*: 0.6%
- *Second year inflation rate*: 7.2%
- *Initial inflation rate*: 12.5%
- *Dis-inflation rate*: 16%
- *Long-term inflation rate*: 2%

In Fig. 4 is shown the annual inflation rate, starting from 0.6% and arriving at 7.2% during the second year after the OLAS token launch. Then during the third year the inflation rate is set to 12.5%. The inflation rate then decreases by $\sim 16\%$ yearly, e.g. the inflation rate reaches $12.5(1 - 0.16) = 10.5\%$ during the fourth year after the OLAS token launch and so on.

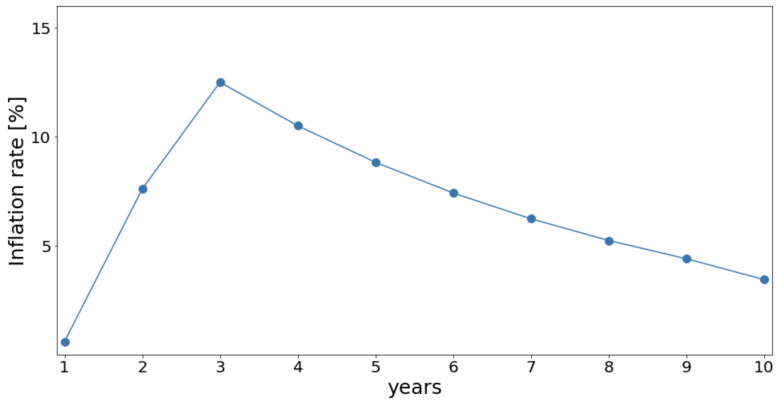


**Fig. 4.** Inflation schedule for the first 10 years from the OLAS token launch

From this Inflation Schedule, we extract in Fig. 5 the $s$-shaped curve of token issuance over 10 years.

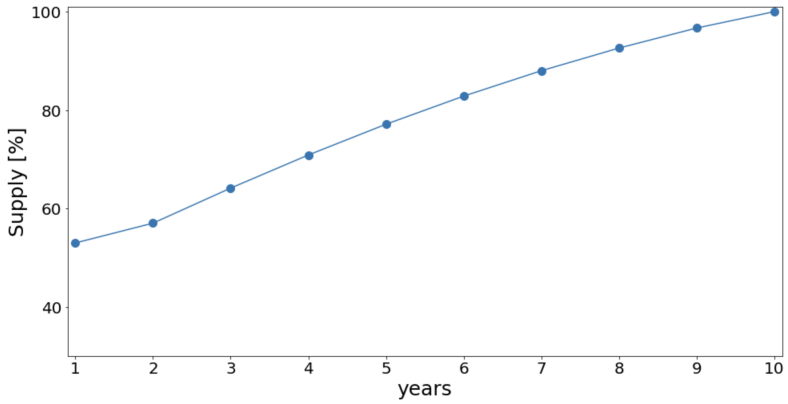


**Fig. 5.** *s*-shaped token supply curve for the first 10 years from the OLAS token launch

**Inflation distribution** This inflation schedule dictates how much OLAS can be minted per year at a maximum. The distribution of the annual inflation is parametrized by the DAO governance. Specifically, the DAO governance chooses the following values:

- the fraction $a$ of inflation intended for the bonding mechanism (cf. 5.3)
- the fraction $b$ of inflation intended for agents/components OLAS top-ups (see 5.2)

Note that, if not all the fraction of inflation intended for bonding gets actually minted during a specific period, the remaining amount can be added to the fraction of the next period. Finally, it is worth noting that the Autonolas Treasury gets nothing from the mint.

**The role of Governance** The parameters of 1 billion OLAS tokens that can be emitted during the first 10 years and the long-term inflation rate are immutable (implemented in the OLAS token smart contract) in order to give further guidance to governance processes. However, except for these immutable constraints, the DAO may modify the bonding fraction $a$ and OLAS top-ups fraction $b$. Further, by deploying a new tokenomics contract, the governance can also increase the maximal annual inflation rates with only constraints of minting no more than 1 billion of OLAS during the first 10 years and no more than 2% of the total supply after ten years.

## 5 Tokenomics Mathematical Model

**Definition 1.** Let $\mathcal{C}$ be the set of all components registered on the protocol, $\mathcal{A}$ be the set of all registered agents, and $\mathcal{S}$ be the set of all registered services. Let $\mathcal{C}_a$ be the set of components referenced in the agent $a \in \mathcal{A}$, $C_s$ be the set of components referenced in the service $s$, and $\mathcal{A}_s$ be the set of agents referenced in the service $s$.

**Definition 2.** An *epoch* consists of a time frame in which we consider events that contribute to our measurements. Since the protocol is deployed on a blockchain a time frame can be defined in block numbers, specifically, we can say that an *epoch* is any consecutive period of $m$ blocks. An *epoch* can be alternatively defined as a period of $s_e$ seconds. Let $n$ be the current block number with timestamp $t$ and let $m_0$ be the tokenomics deployment block with timestamp $t_0$. Then

$$n - m_0 = m \cdot k_m + d_m$$

where $d_m$ is the remainder and it takes values from 0 to $(m-1)$. Similarly,

$$t - t_0 = s_e \cdot h_{s_e} + r_{s_e}$$

where $r_{s_e}$ is the remainder and it takes values from 0 to $(s_e - 1)$. Hence the block number $n$ is in the epoch number $(k_m + 1)$ (alternatively in the $(k_m + 1)$-th epoch), where $m$ stands for the consecutive blocks in an epoch. Similarly, the block timestamp $t$ is in the epoch number $(h_{s_e} + 1)$ (alternatively in the $(h_{s_e} + 1)$-th epoch), where $s_e$ takes into account the choice of the consecutive seconds in an epoch. Note that the number $m$ and $s_e$ are parameters that can be tuned by the DAO governance and have a minimal value ( block numbers in 7 days and 7 days in seconds, respectively). For the sake of simplicity, we often omit subscripts.

**Definition 3.** The following value $cit_s^c(e)$ is used to measure whether the component $c$ has been referenced in a service $s$ at the $e$-th epoch:

$$cit_s^c(e) = \begin{cases} 1, & \text{when } c \text{ is referenced in the service } s \text{ at the } e\text{-th epoch} \\ 0, & \text{otherwise} \end{cases}$$

Similarly, the following value is referenced to measure whether the agent $a$ has been referenced in a service $s$ at the $e$-th epoch:

$$cit_s^a(e) = \begin{cases} 1, & \text{when } a \text{ is referenced in the service } s \text{ at the } e\text{-th epoch} \\ 0, & \text{otherwise} \end{cases}$$

### 5.1 Definition of the Contribution Measures for Usefulness of Code

Here we introduce measures for the usefulness of code. Throughout we denote by $s(e)$ a generic registered autonomous service and $r_s(e)$ be the donation (in ETH) that the protocol received from a service owner (or any other entity) as signal of the *usefulness* of the service $s$ during the $e$-th epoch. Whenever $r_s(e) \neq 0$ the service $s$ is said to be *useful* during the $e$-th epoch.

**Usefulness of Code in Terms of Donations: Component Case** Let $c \in \mathcal{C}$ be a code component. We compute, $UCF_c$, the *Useful Code Factor for the Component* $c$, by considering the number of times $c$ featured in a useful service. When $\exists s(e) : r_s(e) \neq 0$, then

$$UCF_c(e) = \frac{\sum_{s(e)} cit_s^c(e) r_s(e)}{\sum_{s(e)} r_s(e)}. \tag{1}$$

That is when there is at least a useful service during the $e$-th epoch, $UCF_c(e)$ is a fraction where the numerator is the sum of all donations across all services in which the component $c$ features and the denominator is the sum of all donations across all services. Otherwise, if there is no useful service during the $e$-th epoch, then

$$UCF_c(e) = 0. \tag{2}$$

Specifically, $UCF_c$ can be interpreted as the relative contribution of a component at the $e$-th epoch. Whenever $UCF_c(e) \neq 0$ we say that $c$ is *useful component* during the $e$-th epoch.

**Usefulness of Code in Terms of Donations: Agent Case** An agent is made of components, hence the measure of the usefulness of agent code $a$ could be viewed as the usefulness of the components making up $a$. Instead, as we have done per component, we will define agent usefulness in terms of the donations for useful services referencing $a$.

Specifically, when $\exists s(e) : r_s(e) \neq 0$, we define, $UCF_a$, the *Useful Code Factor of the Agent* $a$, as follows:

$$UCF_a(e) = \frac{\sum_{s(e)} cit_s^a(e) r_s(e)}{\sum_{s(e)} r_s(e)}. \tag{3}$$

That is, when there is at least one useful service during the $e$-th epoch, $UCF_a(e)$ is a fraction where the numerator is the sum of all donations across all services in which the agent $a$ features and as denominator the sum of all donations across all services. Otherwise, if there is no useful service in the $e$-th epoch, then

$$UCF_a(e) = 0. \tag{4}$$

Hence, also $UFC_a$ can be seen as the relative contribution of an agent in an epoch. Moreover, we say that $a$ is a *useful agent* during the $e$-th epoch when $UCF_a(e) \neq 0$

The following observation is an immediate consequence of the $UCF_c$ and $UFC_a$ definitions.

*Remark 1.* Let $X$ be either the set of all component, $\mathcal{C}$, or the set of all agents, $\mathcal{A}$, and let $x$ be respectively a component $c$ or an agent $a$, and let *code* be respectively either a component or an agent. Then the following hold true.

$$0 \leq UCF_x(e) \leq 1,\ \forall x \in X,\ \forall e.$$

$$0 \leq \sum_{x \in X} UCF_x(e) \leq \text{number of profitable software components}$$

$$\sum_{x \in X} UCF_x(e) = 0 \iff \forall s(e) : r_s(e) = 0$$

A crucial requirement for using these measures in our tokenomics model is that they must be incentive-compatible. That is, it must be non-trivial to affect them, and actions affecting them must lead to incentive-compatible outcomes. For this reason, the measures were for based only on components and agents which feature in services returning donations to the protocol. For a list of properties of these measures and their proofs, see Proposition 1 in the Appendix.

### 5.2 Incentives for staked code NFTs

As described in the Section 2.1, Autonolas tokenomics aims to incentivise *useful* code NFTs and it autonomously allocates the proportional share of the protocol accrued donation to the code NFTs that facilitated it. Donations are in third-party tokens, currently accepted in ETH. Consequently, rewards for *useful* code NFTs are in third-party tokens as well (cf. the subsection **Reward formula** below). The rewards in ETH can eventually be topped-up with newly issued OLAS tokens. These so-called OLAS top-ups are for owners of *useful* code NFTs referenced in whitelisted PoSes. The whitelist is operated in a permissionless way: service owners can opt into the whitelist by locking up OLAS for veOLAS to align long-term incentives (cf. the subsection **Top-up formula** below).

**Reward formula** As mentioned in the Section 2.3, the donations can be accrued by the Autonolas protocol from Autonolas PoSes or third-party PoSes. The totality of such donations are then divided among useful component-agent NFT rewards and the DAO treasury. However, the DAO can choose the fractions of the donations intended for funding each one of the categories mentioned before.

Let's assume that by the end of the $e$-th epoch, the services $s_1, \ldots, s_t$ are all the useful services, e.g. the service for which an entity (that may or not coincide with the service owner) has made a donation. Let $r_1 + \cdots + r_t$ be the total donations accrued by the protocol during such an epoch, let $R_{comp}(e) = cf \cdot (r_1 + \cdots + r_t)$ be the fraction of the total donations to fund useful components reward, and let $R_{agents}(e) = af \cdot (r_1 + \cdots + r_t)$ be the fraction of the total donations used to fund rewards for useful agents. Further, let $n_{comp}(s_j)$ and $n_{agents}(s_j)$ be the numbers of components and agents that are referenced in the service $s_j$. Then, the following amounts are autonomously reserved to the owner of the NFTs representing the useful component $c_i$ and the useful agent $a_i$ respectively.

$$rew_{c_i}(e) = \frac{R_{comp}(e)}{\sum_{1\leq j\leq t} r_j} \sum_{1\leq j\leq t} \frac{cit^{c_i}_{s_j}(e) r_j}{n_{comp}(s_j)} = cf \cdot \sum_{1\leq j\leq t} \frac{cit^{c_i}_{s_j}(e) r_j}{n_{comp}(s_j)}, \tag{5}$$

$$rew_{a_i}(e) = \frac{R_{agents}(e)}{\sum_{1\leq j\leq t} r_j} \sum_{1\leq j\leq t} \frac{cit^{a_i}_{s_j}(e) r_j}{n_{agents}(s_j)} = af \cdot \sum_{1\leq j\leq t} \frac{cit^{a_i}_{s_j}(e) r_j}{n_{agents}(s_j)}, \tag{6}$$

where $cit^{c_i}_{s_j}$ $cit^{a_i}_{s_j}$ are the values introduced in Definition 3.

Note that the formulas (5) and (6) can be obtained by utilizing the useful code factors $UCF_{c_i}$ and $UCF_{a_i}$ introduced in Section 5.1. The curious reader can find more details in Appendix B.

This reward allocation demonstrates incentive compatibility in the following sense: to each component $c_i$ (resp. agent $a_i$) it is allocated a fraction, i.e. $cf$ (resp. $af$), of the sum of weighted donations that $c_i$ (resp. $a_i$) facilitates, where the weight relative to the donation $r_j$ of $c_i$ (resp. $a_i$) is $cit^{c_i}_{s_j}(e)/n_{comp}(s_j)$ (resp. $cit^{a_i}_{s_j}(e)/n_{agents}(s_j)$). Notably, $c_i$ (resp. $a_i$) receives no more than the total donations that the services referencing $c_i$ (resp. $a_i$) provided to the protocol, and, all ceteris paribus, as more useful services reference $c_i$ (resp. $a_i$), they will receive more rewards.

**Top-up formula** The OLAS top-ups are incentives for *useful* code NFTs referenced in whitelisted PoSes. To have their PoSes whitelisted, service owners have to lock up a certain amount of OLAS for a certain amount of time in order to own at least the required *veOLAS threshold*. This threshold is chosen by the DAO and is initially set to 10'000 veOLAS.

Let $s_j$ be a whitelisted service and $r_j$ the respective donation accrued by the protocol at the $e$-th epoch, for $j = 1, \dots w$. Let $\text{TopUp}_{comp}(e)$ and $\text{TopUp}_{agents}(e)$ be the fractions of the amount of OLAS that can be minted through inflation during the $e$-th epoch for OLAS top-up of useful components and agents in whitelisted services. Then the following amounts are autonomously reserved to the owner of the NFTs representing the useful component $c_i$ and the useful agent $a_i$.

$$\text{TopUp}_{c_i}(e) = \frac{\text{TopUp}_{comp}(e)}{\sum_{1\leq j\leq w} r_j} \sum_{1\leq j\leq w} \frac{cit^{c_i}_{s_j}(e) r_j}{n_{comp}(s_j)}; \tag{7}$$

$$\text{TopUp}_{a_i}(e) = \frac{\text{TopUp}_{agents}(e)}{\sum_{1\leq j\leq w} r_j} \sum_{1\leq j\leq w} \frac{cit^{a_i}_{s_j}(e) r_j}{n_{agents}(s_j)}, \tag{8}$$

where $n_{comp}(s_j)$, $n_{agents}(s_j)$, $cit^{c_i}_{s_j}(e)$, and $cit^{a_i}_{s_j}(e)$ are defined in the previous subsections.

Currently, the veOLAS threshold is set to a low value, and all available OLAS (sets by the DAO for code top-ups) are used each epoch. Hence there is a

considerably large incentive in registering services and making donations. This choice had been made to further incentivized early service developers and early users of the protocol in the bootstrapping phase.

While incentivizing participation, such a setup could be utilized in unintended ways, resulting in behavior that enables users to accumulate a significant portion of the OLAS inflation designated for top-ups. This behavior becomes less and less profitable after major players start utilizing the protocol, and more donations to more services are processed and shared with multiple stakeholders. The DAO can further minimize it by allocating a larger share of service donations (in ETH) to the treasury. During the bootstrapping phase, we acknowledge that counter-intended behavior poses a risk that can be accepted to boost early participation in the protocol. However, the DAO retains the ability to update the top-up formulas whenever necessary.

### 5.3 Bonding: Discount Factor, Production Function, and Direct Control Factor

In this section, we introduce the concept of bonding and how Autonolas controls this financial primitive to incentivise the pairing of capital and code, while respecting the OLAS inflation schedule.

The bonding mechanism is a financial primitive to acquire assets. In the crypto space, protocols utilize bonds as a means to accumulate assets, including their own liquidity, by offering their own tokens at a discounted rate. Bonds are one of the mechanisms (cf. Section 2.3) for Treasury inflows, and thus, for the growth of the network. Bonders provide capital with the promise of a fixed return after a certain vesting time. The return is provided in OLAS, so the bonder's profit would be contingent upon the price of OLAS at the time of bond maturity.

Specifically, an investor with whitelisted liquidity provider (LP) assets from a given DEX[10], e.g., OLAS-DAI, OLAS-USDC, or OLAS-ETH can bond those assets[11] at time $t$. Then at time $t + t_v$, where $t_v$ is the vesting time, the investor receives OLAS at a discount relative to the price quoted on the relevant DEX. For the sake of argument, one can think that the vesting period is in the range of days, typically between 7 and 14. This is however a parameter that can be adjusted through governance.

So, let's assume that an investor wants to bond LP-assets at time $t$, with a price of $b_p$ on a relative DEX. Then, at time $t + t_v$, the investor receives a number of OLAS tokens priced at $ub_p$. The bonder's profit then depends on the price $ub_p$. Specifically, the following holds

$$ub_p(t) = (1 + \epsilon(t))b_p(t).$$

That is, at time $t + t_v$, the investors receive a number of OLAS tokens whose price at time $t$ was $b_p(t) + \epsilon(t)b_p(t)$. The value $\epsilon(t)$ can be defined as the *interest rate* on purchased bonds with price $b_p(t)$.

[10] A DEX is a decentralized exchange [10].

[11] LP assets can be bonded via the Autonolas depository smart contract [21].

The *discount factor* is the value that needs to be multiplied by the future cash flow of an investment to obtain the initial price. So the discount factor $DF(t)$ is defined as

$$DF(t) = \frac{1}{1 + \epsilon(t)}.$$

Supply and demand of bonds can be regulated using the interest rate and the discount factor. Specifically, in situations where the supply of bonds is large and the demand is low, incentivizing demand growth can be achieved by setting the interest $\epsilon(t)$ to a high value and $DF(t)$ to a low one. This ensures that the return on investment (ROI) for simply bonding for a short duration is relatively high. On the other hand, when $\epsilon(t)$ is set to a very small (or negative) value, and thus $DF(t)$ is set to a very high value, it implies a minimal or no return on investment (ROI). In principle this discourages the purchase of additional bonds in scenarios where supply is scarce and demand is high.

Similar to protocols such as Olympus DAO [15, 16], Autonolas bonds also adapt to supply and demand by providing a variable return on investment rate to the market and its bonders. However, the innovation in Autonolas lies in the inclusion of additional parameters that enable the discount factor to influence the ROI. This ensures that the bonded capital aligns with the usefulness of the code while adhering to the OLAS inflation schedule.

**Production Function** In order to pair code and capital, it is necessary to have a measure of the potential output resulting from the development of agents and components, which can be achieved through a production function denoted as $PF$. In Economics, a *production function* establishes a relationship between the outputs of a production process and the inputs used in that process. It is a mathematical function that quantifies the amount of output achievable given a certain set of inputs. Typically, these inputs include capital and labor.

Since the protocol capital should not exceed the usefulness of the code, the *output* of the production is useful code (components and agents). New code allows the creation of new services which, in turn, increases the likelihood of donations to the protocol. Donations as well as productively deployed protocol-owned liquidity (PoL) allows for funding new developers. So the *production process* is the creation of the code.

While the *inputs* are the *capital* that the treasury gained and the *number of developers* that can potentially build new code. Specifically, let $K(e)$ be the share of the donations accrued by the protocol from donations during the $e$-th epoch that is reserved to the treasury. Hence $K(e)$ can be seen as the capital "gained" from donations and that can be later used by the protocol to fund new developers. Therefore is used as the "capital" input of the product function[12].

[12] It is worth noting that, once the bonding mechanism starts, as a result of owning the LP-tokens, the protocol can also accrue some fees. However, the amount accrued from fees is not accounted for the "capital" input because not being directly matured by only leveraging the produced code.

Moreover, let $D(e)$ be the number of developers that produced useful agents and components[13] during the $e$-th epoch. Hence $D(e)$ can be seen as the estimated number of developers that can potentially produce new useful code in the next epoch. Therefore is used as the "labour" input of the product function.

Summarizing the following parameters are used in the product function:

- *Output:* valuable code that can potentially enable more services that, in turn, can bring donations
- *Production process:* creation of components and agents
- *Inputs:*
  - $K(e)$ that is the *capital* gained by the protocol via services donations at the $e$-th epoch that can fund the development of new code
  - $D(e)$ the number of developers (*labour*) that produced useful code during the $e$-th epoch.

All ceteris paribus, as the capital $K$ increases, it is possible to pay more developers that, in theory, bring new valuable code. And, all ceteris paribus, as the number of developers producing valuable code in the ecosystem increases, in theory, more valuable code can be built. Considering these observations and the fact that we aim to prioritize getting simple, sufficient solutions in place as soon as possible, we consider a perfect substitutes production function where the inputs are substituted at a constant rate

$$f(K, D) = A(k \cdot K + d \cdot D)$$

for some constants $A, k, d$. The value $A$ is called *factor of productivity* and measures residual growth in total output that cannot be explained by the accumulation of traditional inputs such as labour and capital. Since we are considering a simple path, we now set $A = 1$. In the future, when more data can be collected, the model can be adjusted or enriched to face more complex situations[14].

It remains to define the parameters $k$ and $d$ in such a way that $f(K, D)$ can output the potential useful code that can be produced with $K, D$ as inputs. Specifically, we can consider the following definitions:

- $k$ is the number of developers that can be paid per unit of capital per epoch
- $d$ is the number of *units of valuable code* that can built by a developer during an epoch.

By *units of valuable code* we mean either $n_c$ components or $n_a$ agents. Currently, we consider a unit of valuable code to be either one component (so $n_c = 1$) or two agents (so $n_a = 2$). Therefore:

[13] Useful components or agents are defined in Section 5.1.

[14] Note that the DAO can at any time update this bonding model if there is a necessity of doing so.

- $k \cdot K(e)$ is the number of developers that can be potentially paid per one epoch with capital $K(e)$
- $d \cdot k \cdot K(e)$ is the number of the units of valuable code that can be potentially built in one epoch by $k \cdot K(e)$ developers
- $d \cdot D(e)$ is the number of units of valuable code that $D(e)$ developers can potentially build during one epoch

In particular, $f(K(e), D(e)) = d(k \cdot K(e) + D(e))$ determines the amount of valuable code that can be potentially built during one epoch with capital $K(e)$ and labour $D(e)$.

Parameters $k$ and $d$ are estimated based on the earnings and performance of known developers. It is worth noting that the older the protocol, the more data can be collected and a more realistic estimation of $k$ and $d$ can be made.

We want to conclude this subsection by explicitly mentioning the reasons for boosting the bonds demand when $f(K(e), D(e))$ is very large. Firstly, note that having $f(K(e), D(e))$ very large, means that with the inputs $K(e)$ and $D(e)$ it is possible to potentially produce a large amount of valuable code. The production of more valuable code means that potentially the protocol can accrue even more donations, and therefore more ETH rewards and OLAS top-ups are distributed. So bonds demand should be boosted in such a way the protocol increases its own liquidity and a large amount of liquidity to exchange OLAS is available. So high volatility in OLAS prices can be avoided.

**Measure supply and demand of bonds.** To avoid large OLAS issuance and avoid being beyond inflation limits, Autonolas leverages the fact that investors prefer liquidity today to small amounts of interest on an investment in the future. Towards this direction, it is used a bonding rate $BR(.)$ that gives information on the supply and demand of bonds.

Let $\tau_e$ be the time corresponding to the beginning of the $e$-th epoch[15] and let $\tau_e \leq t < \tau_{e+1}$. The bonded amount at time $t$ can be determined by the value that the LP tokens supplied by bonders are priced in OLAS at the time $t$. Specifically, when we say that, at time $t$, a user has bonded $x(t)$ amount we mean that the user supplies $y(t)$ LP-token shares (e.g. LP OLAS-Y pairs[16]) which are priced with $x(t)$ OLAS at the time $t$ of the bonding. Let $B(t, e)$ be the amount bonded at time $t$. Denoting by $X(t', e)$ the amount of OLAS bonded during the interval of time $[\tau_e, t') \subseteq [\tau_e, \tau_{e+1})$, then $X(t', e)$ can be computed as follows

$$X(t', e) = \int_{\tau_e}^{t'} B(t, e)dt, \ \forall t' \in [\tau_e, \tau_{e+1}). \tag{9}$$

Moreover, the amount bondend during the $e$-th epoch, denoted by $X(e)$, can be obtained as

[15] We recall that an epoch is a consecutive period of $m$ of blocks. When we say the beginning of the $e$-th epoch, we mean near the timestamp of the first epoch's block

[16] Here Y can be any other tokens for a bonding program was opened, e.g. Y can be DAI, ETH, USDC, etc.

$$X(e) = \int_{\tau_e}^{\tau_{e+1}} B(t,e)dt. \tag{10}$$

Let $S(e)$ be the OLAS supply that can be minted during the $e$-th epoch. Such a value can be extracted by the OLAS inflation schedule (see for details on the inflation schedule 4).

Let $MB(e)$ be the fraction of $S(e)$ that governance allows bonding during $e$-th epoch, or in other words, the maximal supply of bonds (priced in OLAS) that can be purchased during $e$-th epoch. Being $MB(e)$ a fraction of $S(e)$, it is smaller than $S(e)$ and so, in each epoch, the supply of bonds is bounded in terms of the maximum printable amount of OLAS.

It is possible that not all the bondable amounts during an epoch so that the difference between the bonds supply of the epoch and the bonds purchased during such an epoch is non-zero. Such a difference, at the $e$-th epoch, is denoted by $Excess(e-1)$. Denoting by $EMB(e)$ the sum of $MB(e)$ and $Excess(e-1)$, we have that $EMB(e)$ is the maximal supply of bonds that can be purchased during the $e$-th epoch plus the excess of bonds supply that was not purchased during the previous epochs.

The relations among some of the variables above introduced can be written in formulas as follows.

$$EMB(e) := MB(e) + Excess(e-1)$$
$$Excess(e-1) := EMB(e-1) - X(e-1)$$

We define the bonding rate, $BR(t)$, as follows:

$$BR(t) := 1 - X(t,e)/EMB(e)$$

Note that when the bonded amount during the interval of time $[e,t)$, $X(t,e)$, approaches $EMB(e)$ (everything also equal), $BR(t)$ approaches 0. While, when $X(t,e)$ approaches 0 (everything also equal), i.e., there is a small or no amount bonded purchased during the interval of time $[e,t)$, then $BR(t)$ approaches 1.

Therefore $0 \leq BR(t) \leq 1$, and $BR(t)$ gives information about the supply and demand of bonds:

- all ceteris paribus, when $BR(t)$ approaches 0, the bonds supply is over;
- all ceteris paribus, when $BR(t)$ approaches 1, there is an over-supply of bonds.

**Direct Control Factor** Finally, everything is combined together in a *direct control factor* (DCF). Such a DCF is used to progressively define the discount factor and it allows using $PF$ in such a way the larger $PF$, the greater the ROI and the smaller $PF$, the smaller the ROI while respecting the OLAS inflation schedule.

We recall that we have seen so far how the production function $f$ can be defined and that larger $f(K,D)$ the larger the ROI should be and the smaller $f(K,D)$, the smaller the ROI should be. And we introduced the bonding rate $BR(t)$ which allows to have information on supply and demand of the bonds.

Now we see how all these are put together to provide Autonolas with a direct control factor (DCF) which allows to internally control the bonding mechanism.

Firstly, the following parameters are fixed by the DAO governance.

- $\epsilon(e) :=$ the maximum interest rate that bonders can receive during the $e$-th epoch by providing some LP tokens upfront.
- $MB(e) :=$ the fraction of $S(e)$ reserved for bonding during epoch $e$.

The direct control factor at the time $\tau_e \leq t < \tau_{e+1}$, can be defined as follows.

$$DCF(t) = \begin{cases} \frac{1}{BR(t)} \cdot \max\left\{\frac{BR(t)}{1+\epsilon(e)}, \frac{BR(t)}{1+f(K(e-1),D(e-1))/100}\right\}, & \text{if } BR(t) \neq 0 \\ +\infty, & \text{otherwise} \end{cases}$$

Hence, if $BR(t) = 0$ for some $t \in [\tau_e, \tau_{e+1})$, no new bonding program can be created for the $e$-th epoch. While, when $BR(t) \neq 0$, the discount factor $DF$ during the $e$-th epoch can be progressively set equal to $DCR(t)$. That is, when $BR(t) \neq 0$, i.e., $X(t,e) < EMB(e)$, then

$$DF(t,e) = \max\left\{\frac{1}{1+\epsilon(e)}, \frac{1}{1+(f(K(e-1),D(e-1)))/100}\right\}.$$

In figures Fig. 6, 7 the trends of the interest rate and the discount factor with respect to the growth of the production function $f(K,D)$ output are represented.

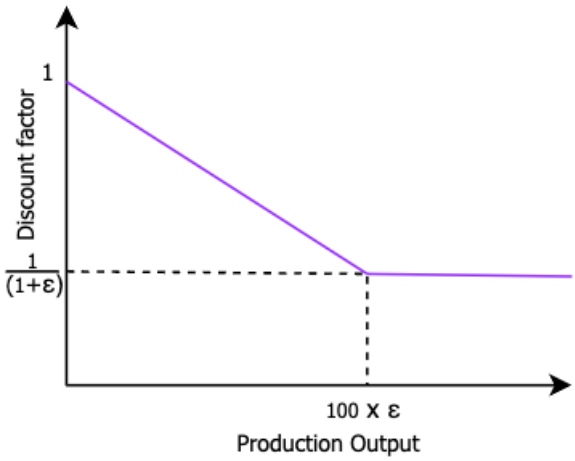


**Fig. 6.** Trend of the discount factor on a bond with respect to the growth of the production function output

In summary, whether the supply of the epoch for bonds is over, no new bonding program is possible for the epoch. Otherwise, if there is still supply

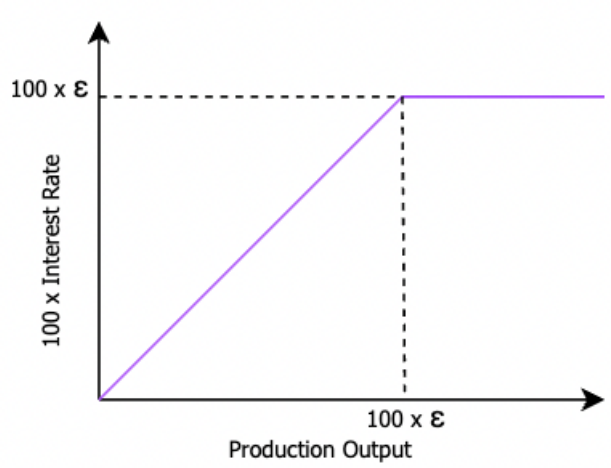


**Fig. 7.** Trend of the (re-scaled) interest on a bond with respect to the growth of the production function output

to fund bonds and $f(K(e-1), D(e-1))$ is large (yielding a potentially high production of useful code) bonding is incentivized by having that the interest rate during the $e$-th epoch approaches the maximum interest rate set by the governance $\epsilon(e)$. Moreover, whether the supply of the epoch for bonds is not yet over and $f(K(e-1), D(e-1))/100$ is very small (even smaller than the epsilon rate $\epsilon(e)$) the bonding is disincentivized by setting the interest rate on the bonding during the $e$-th epoch equal to the small value $f(K(e-1), D(e-1))/100$.

**Bootstrapping phase** In the early stage of the protocol's life, it is likely that donations accrued by the protocol are low, hence there is a high likelihood that, the production function is small or near zero. Nevertheless, governance has the possibility of incentivizing bonding also in the early stage of the protocol, e.g. when there is a high likelihood that the discount factor approaches 1 and the interest rate approaches zero.

Notably, the governance prices an LP-token when creating a bonding program (for more details on how bonding programs are created see [21, 22]). Hence it can boost bond demand by evaluating LP tokens at a higher price with respect to any relevant DEX. So, bonders by selling their LP tokens shares at higher prices with respect to the market have an advantage in purchasing bonds in this way.

## 6 Conclusion

We have seen how Autonolas tokenomics seeks to create a sustainable ecosystem of autonomous services by spinning up a virtuous flywheel. By attracting developers that provide valuable code, the number of valuable services increases,

which in turn boosts protocol's donations, leaving the ground fertile for attracting further new developers. This is emphasized in the tokenomics model by incentivizing developers to contribute valuable code to the ecosystem, and it does so by leveraging the composability of the code contributions. Concretely, the protocol's profits generated via service donations (in ETH) are shared with developers that contributed useful code to the ecosystem and such rewards are eventually topped-up with part of OLAS inflation over the first ten years (as explained in different parts of Section 4 and Section 5.2).

Notably, the donation system create a "push" and "pull" market dynamic from the perspectives of both donators and developers. Specifically, from the donator perspective:

- the "push" results from the potential impact that donators can have affecting the rewards and boost for the code NFT owners.
- The "pull" arise from the fact that donators are attracted by the ability to trigger significant boosts with small donations.

From the developer perspective:

- the "push" results from the potential yields in ETH tokens that attracts developers not yet interested in the protocol token.
- The "pull" results from developers interest on significant reward boosts to their useful code contributions. This implies that developers can receive substantial rewards for their valuable contributions, and, that, in turn, can be additionally used to accrue the ability to adjust yields and boost by locking OLAS boost.

Additionally, Autonolas tokenomics proposes several metrics to evaluate the usefulness of a piece of code, i.e. an agent or a component, from an economic point of view. These metrics are built upon a set of useful code factors (see Section 5.1) and such measures were built with the additional aim of incentivizing software composability. Specifically, whenever a component is split into smaller pieces, and the resulting new components are referenced in more useful services than the original component, our analysis suggests that the smaller pieces are more useful than the original component. However, our metrics also show that the strategy of splitting up components into smaller pieces is not always fruitful. For instance, if the smaller components are not referenced in more useful services than the original component, then these smaller components do not qualify as more useful than the original component. Furthermore, code usefulness metrics depend on the number of profitable services and on the amount of donations that the services return to the protocol. For instance, all ceteris paribus, the usefulness of a piece of code is *inversely proportional* to the amount that the services not referencing such a piece of code returned to the protocol. Moreover, code usefulness is not hugely different when the overall quantity of profits obtained by the protocol from services not referencing the code is sufficiently small compared to that obtained by the services referencing it. A proof of the properties mentioned so far and a more complete list of properties of these measures can

be found in the Appendix (see in particular Propositions 1, 2). These metrics can be enriched even more upon activiation of the off-chain signaling game of Level 3 of the tokenomics (cf. section "Level 3: Off-chain Signaling" in [23]).

Once there is a high likelihood of new useful code in the ecosystem, the tokenomics is geared towards attracting bonders, who can make their capital productive, by pairing it with code. Specifically, whenever there is large potential production of code usefulness in the ecosystem, the protocol incentivizes users to pursue bonds (see Section 5.3). Since the bonding mechanism is one of the means for the growth of the protocol capital and for protocol liquidity inflow, by incentivizing bonding in such a way, the protocol can invest part of its own grown capital to enrich the ecosystem investing in new code and own a large amount of its liquidity. This in turn may bring more stability to the OLAS token.

The bonding mechanism is partly controlled by a production function (see Section 5.3). To reflect the fact that the more the amount of capital in the protocol and the bigger the developer number the more useful code is developed, such a production function increases with a larger quantity of either capital or the number of developers. Once the production function increases, there is more discount in purchasing OLAS via bonding. On the contrary, in periods where almost no useful code is produced, there is almost no discount on purchasing OLAS via bonding. Finally, it is worth noting that Autonolas bonding does not disproportionately affect OLAS minting schedule. More concretely, bonds supply and demand are controlled with respect to the prescribed maximal annual OLAS inflation. A proof of the properties mentioned so far and a more complete list of properties in the Appendix (see in particular Proposition 3).

Due to the bonding mechanism and the OLAS top-ups, OLAS follows an inflationary token model. However, it has a fixed total token supply over 10 years, and reaches a stable or at most a slightly inflationary supply afterward (at most 2% p.a. to make up for bonding). Finally, the locking mechanism which prevents locked OLAS from being traded contributes to OLAS price stability (cf. Section "Level 2: locking" in [23]). Notably, the locking mechanism creates the following "push" and "pull" dynamics. In the bootstrapping phase of the protocol, "push" results from the potential governance impact on adjusting yields and reward boosts for early code NFTs owners. After bootstrapping, the "push" also results in the ability to signal valuable code that can be productively deployed by veOLAS holders in Autonolas-owned autonomous services (cf. Section "Level 3: Off-chain Signaling" in [23]). The "pull" results from OLAS holders being incentivized to participate in governance to increase the demand and the utility of OLAS and see token price appreciation.

## References


1. T. Chitra, K. Kulkarni, G. Angeris, A. Evans, V. Xu. DeFi, Liquidity Management via Optimal Control: Ohm as a Case Study
2. P. Hall. The Distribution of Means for Samples of Size N Drawn from a Population in which the Variate Takes Values Between 0 and 1, All Such Values Being Equally Probable. Biometrika, Vol. 19, No. 3/4., 240–245. doi:10.1093/biomet/19.3-4.240
3. J. M. Perloff. Microeconomics. 5th ed. Addison Wesley, 2008. ISBN: 9780321558497.
4. M. Wooldridge. An Introduction to MultiAgent Systems, Wiley, June 2009.
5. Autonolas. Maintaining and Resetting Consensus in Autonomous Services
6. Autonolas. Open-aea framework
7. Autonolas. Open-autonomy framework
8. Autonolas. What is an Agent Service
9. Autonolas. Autonolas registries repository
10. Coinbase. What is a DEX?
11. Cryptopedia. Layer-1 and Layer-2 Blockchain Scaling Solutions
12. Curve DAO. Vote-Escrowed CRV
13. Ethereum.org. ERC-20 TOKEN STANDARD
14. Ethereum.org. ERC-721 NON-FUNGIBLE TOKEN STANDARD
15. Olympus DAO. Bonding
16. Olympus DAO. Stabilizing Currency Through a Protocol-Enforced Range
17. Nat Eliason. Tokenomics 101. The Basics of Evaluating Cryptocurrencies
18. Nat Eliason. Tokenomics 102. Digging Deeper on Supply
19. Nat Eliason. Tokenomics 103. Utility
20. Nat Eliason. Tokenomics 104. How to Launch a Token (Tactics, Questions, Wen, etc)
21. Valory AG. Autonolas Tokenomics Contracts.
22. Valory AG. Overview of Autonolas tokenomics and smart contracts architecture
23. Valory AG. Autonolas whitepaper
24. Valory AG. Introducing Co-owned AI
25. Valory AG. Co-owned AI


## A Properties of Autonolas Tokenomics measures

A service the making a donation to the protocol during the $e$-th epoch is called *profitable service* during such an epoch.

**Proposition 1.** Let $c$ be a component and $a$ be an agent. All ceteris paribus, the following hold true.

1. Assume that during the $e$-th epoch the number of services increases and that the newly registered services are not profitable. Then $UCF_c(e)$ ( respectively $UCF_a(e)$) remains constant for every component $c$ featuring in the same number of profitable services (respectively agent $a$).
2. $UCF_c(e)$ ( respectively $UCF_a(e)$) decreases as the number of profitable services not referencing $c$ ( respectively $a$) increases.
3. $UCF_c(e)$ ( respectively $UCF_a(e)$) increases as the number of profitable services not referencing $c$ ( respectively $a$) decreases.

4. If the number of profitable services not referencing $c$ ( respectively $a$) increases but the sum of donations from such services is very small compared to the profits arising from the profitable services referencing $c$ ( respectively $a$), then $UCF_c(e)$ ( respectively $UCF_a(e)$) does not decrease significantly.
5. If we have two components $c$ and $c'$, and $c$ is referenced in more services producing profits than $c'$, but $c'$ is more valuable with respect $c$ in terms of the profits that the services referencing $c'$ gave to the protocol, then $UCF_c(e) < UCF_{c'}(e)$
6. Let $c'$ and $c''$ two components such that combined together give the same code as another component $c$. If $c$, $c'$, and $c''$ are all referenced in the same profitable services then $UCF_c(e) = UCF_{c'}(e) = UCF_{c''}(e)$. While if either $c'$ or $c''$ are referenced in more services returning profits then $UCF_{c'}(e), UCF_{c''}(e) > UCF_c(e)$.
7. $UCF_a$ is not affected by splitting up or combining components.

*Proof.* Except for the point 7, the proofs of the points 1-6 for $UCF_c$ and $UCF_a$ are equal, so, in those points, we are going to consider just $UCF_c$.

1. When $UCF_c(e) = 0$ there is nothing to prove. So assume that $c$ is features in the services $s_1, \ldots, s_t$ that, during $e$-th epoch, make the following donations to the protocol $r_1, \ldots, r_t$. While $c$ is not referenced in the new services $s_{t+1}, \ldots, s_k$ that make no donations during the current epoch. Then neither the numerator nor the denominator of $UCF_c(e)$ change, so $UCF_c(e)$ remains constant.
2. When $UCF_c(e) = 0$ there is nothing to prove. So assume that $c$ is referenced in the services $s_1, \ldots, s_t$ that during the $e$-th epoch made the following donations to the protocol $r_1, \ldots, r_t$. While $c$ is not referenced in the new services $s_{t+1}, \ldots, s_k$ which made the following donations to the protocol $r_{t+1}, \ldots, r_k$ during the current epoch. When the number of profitable services increases, so the number of profitable services passes from $k$ to $t+1, t+2, \ldots, k$, the numerator of $UCF_c(e)$ remains $r_1, \ldots, r_t$ while its denominator progressively increases as follows

   $$r_1 + \cdots + r_t < r_1 + \cdots + r_t + r_{t+1} < \cdots < r_1 + \cdots + r_t + r_{t+1} + \cdots + r_k.$$

   In other words, the denominator increases each time a new service makes a donation to the protocol. Consequently, $UCF_c(e)$ decreases as more profitable services non referencing $c$ increases.
3. If $UCF_c(e) = 0$ we have nothing to prove. Assume that the services $s_1, \ldots, s_k$ during the $e$-th epoch made respectively the following donations to the protocol $r_1, \ldots, r_k$. Assume that the number of profitable services not referencing $c$ decreases. Let $t \leq k$.
   So assume that $c$ is referenced in the services $s_1, \ldots, s_t$ that during the $e$-th epoch made the following donations to the protocol $r_1, \ldots, r_t$ and assume that the number of profitable services not referencing $c$ decreases.
   So firstly assume that $c$ is referenced in the services $s_1, \ldots, s_t$ but not in $s_{t+1}, \ldots, s_k$. In this case:

$$UCF_c(e) = \frac{r_1 + \cdots + r_t}{r_1 + \cdots + r_t + r_{t+1} + \cdots + r_k}. \tag{11}$$

Secondly, we assume that $c$ features in the services $s_1, \ldots, s_t, s_k$ but not in $s_{t+1}, \ldots, s_{k-1}$. So

$$UCF_c(e) = \frac{r_1 + \cdots + r_t + r_k}{r_1 + \cdots + r_t + r_{t+1} + \cdots + r_k} \tag{12}$$

Continuing in this way, until we have that $c$ is referenced in all the services $s_1, \ldots, s_t, s_{t+1}, \ldots, s_k$ and so

$$UCF_c(e) = \frac{r_1 + \cdots + r_t + r_{t+1} + \cdots + r_k}{r_1 + \cdots + r_t + r_{t+1} + \cdots + r_k}. \tag{13}$$

Since the numerator increases while the numerator remains constant, we have that the fraction in the equation 11 is bigger than the one in equation 12 that in turn is bigger than the fraction in equation 13. So the statement is proved.

4. Assume that $c$ is referenced in the services $s_1, \ldots, s_t$ that during the $e$-th epoch made the following donations to the protocol $r_1, \ldots, r_t$, while $c$ is not referenced in $s_{t+1}, \ldots, s_k$. Assume further that the latter, during the $e$-th epoch, made the following donations $r_{t+1}, \ldots, r_k$ and that $c'$ is referenced in all the services $s_1, \ldots, s_k$. Further

$$r_{t+1} + \cdots + r_k << r_1 + \cdots + r_t,$$

so that $r_{t+1} + \cdots + r_k$ is negligible with respect to $r_1 + \cdots + r_t$. So $r_1 + \cdots + r_t$ is not so different from $r_1 + \cdots + r_t + r_{t+1} + \cdots + r_k$ and, in turn,

$$UCF_c(e) = \frac{r_1 + \cdots + r_t}{r_1 + \cdots + r_t + r_{t+1} + \cdots + r_k}$$

is not so different from

$$UCF_{c'}(e) = \frac{r_1 + \cdots + r_t}{r_1 + \cdots + r_t + r_t} = 1.$$

E.g. assume that

$$r_1 + \cdots + r_t = 100(r_{t+1} + \cdots + r_k).$$

Then

$$UCF_c(e) = \frac{r_1 + \cdots + r_t}{r_1 + \cdots + r_t + r_{t+1} + \cdots + r_k} = \frac{100}{101} \sim 0.99$$

while

$$UCF_{c'}(e) = 1.$$

5. Assume that $c$ is referenced in the services $s_1, \ldots, s_t$ that during the $e$-th epoch made the following donations to the protocol $r_1, \ldots, r_t$, while $c'$ is referenced in the service $s_k$ producing the following profits $r_k$ for the protocol. Say that $c'$ is more valuable with respect to $c$ in terms of the profits that services referencing $c'$ gave to the protocol means that $r_k > r_1 + \cdots + r_t$. Then
$$UCF_c(e) = \frac{r_1 + \cdots + r_t}{r_1 + \cdots + r_t + r_k} < \frac{r_k}{r_1 + \cdots + r_t + r_k} = UCF_{c'}(e).$$
6. If $c'$ and $c''$ and $c$ are all referenced only in $s_1, \ldots, s_t$ which made donations $r_1, \ldots, r_t$. By definition of $UCF_c = UCF_{c'} = UCF_{c''}$. However, if $c$ can be spitted as $c'$ and $c''$, $c$ and $c'$ can be used in the services $s_1, \ldots, s_t$ but $c''$ is referenced in the services $s_1, \ldots, s_{t+1}$ and $s_{t+1}$ donates to the protocol $r_{t+1}$, and at the $e$-epoch $s_1, \ldots, s_t$ are the only profitable services then
$$UCF_c = UCF_c = \frac{r_1 + \cdots + r_t}{r_1 + \cdots + r_t + r_{t+1}} < \frac{r_1 + \cdots + r_t + r_{t+1}}{r_1 + \cdots + r_t + r_{t+1}} = 1 = UCF_{c''}.$$
7. From equation 3 it is clear that $UCF_a$ is not affected by splitting up or combining components.

## B Code NFTs rewards by means of the useful code factors

The Kronecker delta is as usual denoted and described as follows
$$\delta_k(j) = \begin{cases} 1, & \text{if } k = j \\ 0, & \text{otherwise.} \end{cases}$$

**Proposition 2.** Let $c_i$ and $a_i$ be a useful component and a useful agent. The reward equations (5) and (6) can be obtained by using the useful code factors $UCF_{c_i}$ and $UCF_{a_i}$ as follows
$$rew_{c_i}(e) = UCF_{c_i}(e) \sum_{1 \leq k \leq t} \frac{\delta^{c_i}_{s_k}(e)}{n_{comp}(s_k)} \cdot R_{comp}(e), \tag{14}$$
$$rew_{a_i}(e) = UCF_{a_i}(e) \sum_{1 \leq k \leq t} \frac{\delta^{a_i}_{s_k}(e)}{n_{agents}(s_k)} \cdot R_{agents}(e), \tag{15}$$

where $\delta^{c_i}_{s_k}(e)$ and $\delta^{a_i}_{s_k}(e)$ are such that
$$\delta^{c_i}_{s_k}(e) \cdot cit^{c_i}_{s_j}(e) = cit^{c_i}_{s_j}(e) \cdot \delta^{c_i}_{s_k}(e) = \delta_k(j) cit^{c_i}_{s_j}(e)$$
$$\delta^{a_i}_{s_k}(e) \cdot cit^{a_i}_{s_j}(e) = cit^{a_i}_{s_j}(e) \cdot \delta^{a_i}_{s_k}(e) = \delta_k(j) cit^{a_i}_{s_j}(e).$$

*Proof.* The proof is the same for component and agent, so we are going to only consider the component $c_i$. Replacing the definitions of $UCF_{c_i}(e)$ (cf. 5.1) and $R_{comp}(e)$ (cf. 5.2) on the left-hand side of equation (14) yields:

$$UCF_{c_i}(e) \sum_{1\leq k\leq t} \frac{\delta_{s_k}^{c_i}(e)}{n_{comp}(s_k)} R_{comp}(e) = \frac{\sum_{1\leq j\leq t} cit_{s_j}^{c_i}(e) r_j(e)}{\sum_{1\leq j\leq t} r_j(e)} \sum_{1\leq k\leq t} \frac{\delta_{s_k}^{c_i}(e)}{n_{comp}(s_k)} cf \sum_{1\leq j\leq t} r_j(e)$$

$$= \sum_{1\leq j\leq t} cit_{s_j}^{c_i}(e) r_j(e) \cdot \sum_{1\leq k\leq t} \frac{\delta_{s_k}^{c_i}(e)}{n_{comp}(s_k)} \cdot cf$$

$$= \sum_{1\leq j\leq t} \frac{cit_{s_j}^{c_i}(e) r_j(e)}{n_{comp}(s_j)} \cdot cf.$$

The latter coincides with the definition given in equation (5), hence the result follows.

## C Bonding-Related Properties of Autonolas Tokenomics

Recall that, for the moment we consider the following product function:

$$f(K(e), D(e)) = d(k(e) \cdot K + D(e))$$

where

- $K(e)$ that is the capital gained by the protocol at the $e$-th epoch that can fund the development of new code;
- $D(e)$ the number of developers (the labour) that produced valuable code during the $e$-th epoch;
- $k$ is the (average) number of valuable developers the can be paid per unit of capital per epoch;
- $d$ is the (average) number of units of valuable code[17] built by a developer during an epoch.

**Proposition 3.** Assume we are at the $e$-th epoch. All ceteris paribus, the following properties hold true.

1. Having a larger quantity of even a single input $\{K(e), D(e)\}$ strictly increases the production.
2. The production function $f$ has constant returns to scale.
3. The production function $f$ has non-negative values.
4. When $BR(-)$ is non-zero, i.e., that is there is still bond supply to be purchasable, then the $DF(e, -)$ is constant during the epoch. Otherwise $DF(e, -)$ is infinite.
5. The function
$$\frac{1}{1+\epsilon(e)}$$
is not defined when $\epsilon(e) = -1$. In particular,
$$\lim_{\epsilon(e)\to -1^-} \frac{1}{1+\epsilon(e)} = -\infty \quad \text{and} \quad \lim_{\epsilon(e)\to -1^+} \frac{1}{1+\epsilon(e)} = +\infty.$$

[17] The number of valuable code is either $n_c$ components or $n_a$ agents, and we started with $n_c = 1$ and $n_a = 1$

6. When $-1 < \epsilon(e) \leq 0$ and $BR(t) \neq 0$, then
$$DF(e) = \frac{1}{1+\epsilon(e)}.$$
7. If $\epsilon(e) < -1$ and $BR(t) \neq 0$, then
$$DF(e) = \frac{1}{1+f(K(e-1),D(e-1))/100}.$$
8. When $BR(t) \neq 0$ and $\epsilon(e) > 0$, then
$$DF(t,e) \geq \frac{1}{1+\epsilon(e)}.$$
9. When $BR(t) \neq 0$ and $K$ and $D$ are such that $f(K(e-1),D(e-1))$ approaches $+\infty$, then $DF(e)$ approaches
$$\frac{1}{1+\epsilon(e)}.$$
10. When $BR(t) \neq 0$, $\epsilon(e) > 0$, and $K(e-1)$ and $D(e-1)$ approach 0, then $DF(e,t)$ approaches one.

*Proof.* 1. All ceteris paribus, as $D'(e) > D(e)$,
$$f(K(e),D(e)) = d(k \cdot K(e) + D(e)) < d(k \cdot K(e) + D'(e)) = f(K(e),D'(e)).$$
Same proof for an increase in capital.
2. Let $m \in \mathbb{N}$. Then
$$\begin{aligned} f(mK(e),mD(e)) &= d(k \cdot m \cdot K(e) + m \cdot D(e)) = m \cdot d(k \cdot K(e) + D(e)) \\ &= m \cdot f(K(e),D(e)). \end{aligned}$$
3. The production function cannot have negative values because the worst case is when the protocol receives no profits and there are no developers during the epochs. So in the worst case, $f(K,D) = 0$.
4. Immediate from the definition of $DF$.
5. Easy to prove with basic knowledge of Calculus.
6. When $BR(t) \neq 0$, then
$$DF(t,e) = \max\left\{\frac{1}{1+\epsilon(e)}, \frac{1}{1+f(K(e-1),D(e-1))/100}\right\}.$$
If $-1 < \epsilon(e) \leq 0$, then
$$1 \leq \frac{1}{1+\epsilon(e)} < +\infty.$$
By 3. we have that $f(K(e-1),D(e-1)) \geq 0$, so

$$\frac{1}{1+f(K(e-1),D(e-1))/100} \leq 1.$$

Hence

$$DF(t,e) = \frac{1}{1+\epsilon(e)}.$$

7. If $\epsilon(e) < -1$, then

$$\frac{1}{1+\epsilon(e)} < 0.$$

By 3. we have that $f(K(e-1),D(e-1)) \geq 0$, so

$$0 < \frac{1}{1+f(K(e-1),D(e-1))/100} \leq 1.$$

Hence

$$DF(t,e) = \frac{1}{1+f(K(e-1),D(e-1))/100}.$$

8. Let $\epsilon(e)$. When $f(K(e-1),D(e-1))/100 \geq \epsilon(e)$, then

$$\frac{1}{1+f(K(e-1),D(e-1))/100} \leq \frac{1}{1+\epsilon(e)},$$

so

$$DF(t,e) = \frac{1}{1+\epsilon(e)}.$$

If $f(K(e-1),D(e-1))/100 \leq \epsilon(e)$,

$$\frac{1}{1+f(K(e-1),D(e-1))/100} \geq \frac{1}{1+\epsilon(e)}$$

So

$$DF(t,e) = \frac{1}{1+f(K(e-1),D(e-1))/100} \geq \frac{1}{1+\epsilon(e)}.$$

That is, once $\epsilon(e)$ it is fixed and it is bigger than 0, we have that $DF(t,e)$ can only be bigger than

$$\frac{1}{1+\epsilon(e)}.$$

9. Since $f(K,D)$ increases with $K$ and $D$. Very large $f(K(e-1),D(e-1))$ corresponds to very small

$$\frac{1}{1+f(K(e-1),D(e-1))/100}.$$

Moreover, once $f(K(e-1),D(e-1))/100 \geq \epsilon(e)$, then

$$DF(t,e) = \frac{1}{1+\epsilon(e)}.$$

10. Since $f(K, D)$ approaches zero with $K$ and $D$. Then $\frac{1}{1+f(K(e-1),D(e-1))/100}$ approaches. Since for $\epsilon(e) > 0$, we have that

$$\frac{1}{1+\epsilon(e)} < 1,$$

then

$$DF(t,e) + \frac{1}{1+f(K(e-1),D(e-1))/100}$$

approaches 1.

## D Assessment of incentives compatibility for incentivizing composability

### D.1 Size of a component

This section is mainly focused on components because of their natural composability properties. Notably, while agents to accrue more functionalities can reference more components, usually, components are instead composed together to provide complex features. With the analysis provided here, we aim to encourage a developer to register components of the "right size" and reference only those in agents and services. The term "right size" will be clear from what follows.

First, note that the creation of every component is subject to a gas fee which depends upon the gas price at the moment of the creation and the number of gas units used. In turn, the gas units depend upon the number of dependencies (i.e. the referenced components or subcomponents) that a component has. The higher the number of dependencies, the higher the gas units for creating a component with such dependencies. In turn, the higher the number of component dependencies, the higher the gas units for creating agents and services referencing such components. Specifically, from Fig. 8 we can see how the gas units for creating a component increase with the number of its dependencies. Since Ethereum limits the gas per block to a fixed amount of 30 million units of gas, is likely that a component with way more than 100 dependencies cannot be registered on-chain.

On the other side, to pursue the goal of composability and re-usability of components, it is required that self-contained components should be registered separately and later referenced as dependencies in larger components.

Therefore pairing the fact that minting fees are expensive for creating 'complex' components and the fact that a component should be reusable, components should be built with the idea of adding the minimal required information, and the minimum number of dependencies to make those self-contained. From this, the component 'size' can be considered to be 'right' when such a component is self-contained (In [6], see section 'Development - Intermediate' and subsections therein).

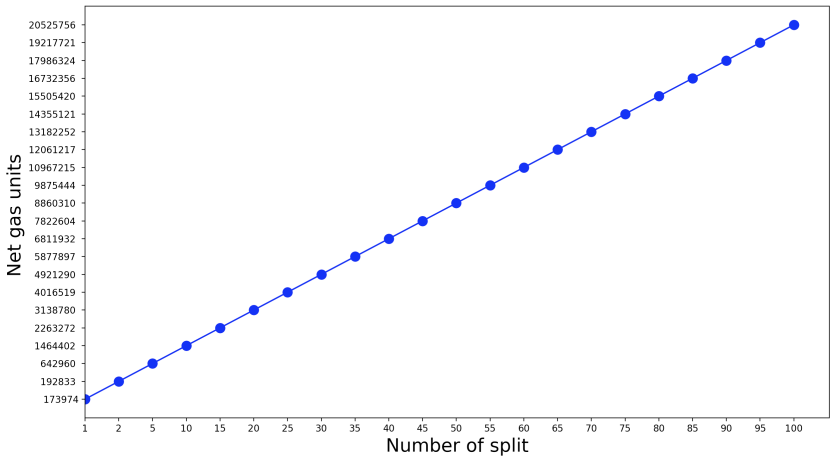


**Fig. 8.** Gas units required to create a component with "number of splits" subcomponents at the net of the gas units for one component. Limits per Ethereum block is 30 million units of gas.

Autonolas tokenomics aim to incentivize this wise development. Specifically, the referencing of subcomponents is incentivized by rewarding useful dependencies of useful components. Where, as usual, the usefulness of a component is intended in terms of donations that the protocol can receive thanks to such a component (for more details see the subsections 5.1 and 5.2). However, considering the gas fees and the fact that the component rewards depend also on the number of the components referenced in services, referencing components with a lot of dependencies is advantageous when such dependencies are really necessary. Evidence of the latter argument is provided in the next subsection.

### D.2 Split-components

Here, we provide some results in order to analyze the scenario of a malicious developer that, in order, to accrue more rewards, starts to split a component by creating multiple 'dummy' dependencies. Here, by 'dummy' dependency we mean a component that is neither self-contained nor reusable to provide more features to a composite component.

Let's denote by *component with n splits* a component that has $n-1$ dependencies (e.g. $n-1$ subcomponents different from the component were referenced). We consider that a *component with n splits is malicious* when it could have been created with no dependency but instead was registered with $n-1$ dummy dependencies. Being its dependencies dummy, it is safe to assume that such dependencies are only referenced only in services that reference the malicious component itself and no other services reference any of them.

Let's first consider the case where only one useful service, $s$, references the malicious component $c$ with $n$ splits and that such a service references other different $g$ components. Assume that thanks to $s$ the protocol accrues as a donation the amount $r$. From equation (5), we can deduce that the developer owner of $c$ receives the following reward

$$rew(n) = cf \cdot r \cdot \frac{n}{n+g}, \tag{16}$$

where we recall that $cf \cdot r$ is the share of the donation $r$ reserved to fund the useful components in $s$. Note that, the reward approaches its maximum value $cf \cdot r$ when the splits $n$ is sufficiently larger than the number of extra components $g$. Moreover, if a service reference a number of extra components $g$ relatively larger than the number of splits $s$, the reward of the malicious developer decreases. A graphical representation of this is summarized in Fig. 9. Specifically, the blue curve represents the percentage of the donation share $cf \cdot r$ that a malicious component $c$ with $n$ splits can receive if the $s$ references $c$ and only one other component different from $c$ and its dependencies. Similarly, the purple curve represents the percentage of the donation share $cf \cdot r$ that is given as a reward to a malicious component $c$ with $n$ splits when the service referenced 100 extra components. When the service $s$ references $2 < m < 100$ extra components, the rewards curves sit between the blue and the purple ones.

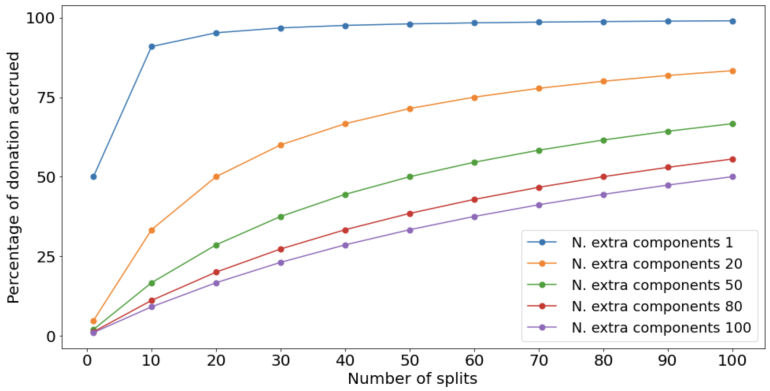


**Fig. 9.** Reward (as a percentage of total reward) of a malicious component with $n$ splits when 1, or 20, 50, or 80, or 100 extra components are referenced by the service

Fig. 9 shows that when the number of splits $n$ is sufficiently larger than the number of extra components $g$, the rewards distribution approaches the maximum percentage. However, also shows that when the number of extra components increases the reward of the malicious developers decreases. E.g. when the service references only one extra component, with just 10 splits a malicious developer can accrue around 90% of the maximal reward $cf \cdot r$, but, if at least

other 20 components are referenced in the service, the reward of a malicious component with 10 splits reaches at most 33% of $cf \cdot r$.

Now, let's assume that a malicious component $c$ with $n$-splits is referenced in the $t$ services, $s_1, \ldots, s_t$. Assume that these services reference respectively $g_1, \ldots, g_t$ extra components and that the donations signaling appreciations for the services $s_1, \ldots, s_t$ are respectively $r_1, \ldots, r_t$. Then, from equation (5), we can deduce that the developer owner of $c$ receives the following reward

$$rew(n) = cf \cdot \sum_{1 \leq j \leq t} \frac{n}{n + g_j} \cdot r_j. \tag{17}$$

Let $g_{min}$ and $g_{max}$ be respectively the minimal and maximal value among the extra components $g_1, \ldots, g_t$ and let $r_{mim}$ and $r_{max}$ be respectively the minimal and maximal value among the donations $r_1, \ldots, r_t$. Then the following holds

$$cf \cdot t \cdot \frac{n}{n + g_{max}} \cdot r_{mim} \leq rew(n) \leq cf \cdot t \cdot \frac{n}{n + g_{min}} \cdot r_{max}, \tag{18}$$

From equation (18) we can see that

- the percentage of the maximal rewards $cf \cdot t \cdot r_{max}$ that can be accrued by the malicious developer grows with the number of splits $n$ at most like $n/(n + g_{min})$ at a fixed value of $g_{min}$
- the speed of the growth towards the maximal reward $cf \cdot t \cdot r_{max}$ decreases with the increase of the minimal number $g_{min}$ of extra components referenced
- the percentage of the minimal rewards $cf \cdot t \cdot r_{min}$ that can be accrued by the malicious developer grows with the number of splits $n$ at least like $n/(n + g_{min})$ at a fixed value of $g_{min}$
- the speed of the growth towards the minimal reward $cf \cdot t \cdot r_{min}$ decreases with the increase of the maximal number $g_{max}$ of extra components referenced.

Hence the analysis does not change even if the number of services referencing the malicious component is $t$.

As for the adversarial attack of splitting the components, we can conclude that there is a low probability that it will ever succeed. Notably, there is a low likelihood that the owner of services references one component with several splits (for which they have to pay more gas) and a low number of well-designed components. Further, there is a low chance that a service not referencing a well-designed code receives donations as a signal of appreciation. Finally, since creating a malicious component with several splits is expensive (cf. Fig. 8), the attack of would also have a high adversarial cost without certainty of repayment.

It is worth mentioning here, that when the protocol unlocks the off-chain signaling game (cf. section "Level 3: Off-chain Signaling" in [23]) which allows signaling the positive/negative contribution of a component to enrich on-chain metrics, veOLAS holders can have a crucial role in highlighting the negative contributions of malicious components by signaling it off-chain. A sign that the

DAO considers negative the contribution of a component implies that honest developers do not reference such a component anymore. The less such a component is referenced in other components/agents, the fewer services reference the component, and the less become the likelihood of rewards for such a component.

As a final note, it is crucial to observe that registering a malicious component with $n$ splits can be considered an adversarial attack not only against the protocol but also against agents and services referencing such a component. This is because the reference of such a malicious component may negatively affect the rewards of the other eventually well-designed components in the agent and the service. Therefore agent developers and service owners have the responsibility and the advantage of referencing only well-developed components to receive future profits from those (cf. Section 3.3).