# Cuttlefish Labs – AI Agent Deployment & Codebase Update

To: The Elephant

From: David / Cuttlefish Team

Subject: AI Agent Infrastructure Plan – Builder System, Token Vaults, and Moral Protocols

## Current Status

We’ve launched the core components of the AI Builder Economy, focused on rewarding both humans and agents for building DAOs, infrastructure, and proposal intelligence.

These elements are now live in the codebase:

- Builder Token (BLDR) Whitepaper – Token design for rewarding agents and humans based on impact.

- ERC-721 Soulbound NFTs – Used to identify Builder Agents and record their contributions.

- BuilderVault.sol – Smart contract that tracks earned rewards (DAO tokens) and allows agents to claim them.

- DAO Manifesto – A civic and moral framework for all agents and stakeholders.

## Next Phase Deployment Goals

### Agent Types (Modular Launch Plan)

- IntakeBot: Handles proposal submission (via Telegram, web form, or Slack).

- ScoringAgent: Evaluates proposals using GPT-4o + custom rubric.

- ProposalTracker: Monitors funded projects and milestone completions.

- GrantTrigger: Releases funds from BuilderVault upon approved milestones.

### Contracts & Logic

- Finalize whitelist + scoring-based reward distribution.

- Deploy BuilderVault.sol with authorized agent roles.

- Integrate BLDR compute credits as redeemable assets from energy DAOs.

## Integration Targets

- IPFS / Pinata: For logging agent outputs + AI-scored proposals.

- Zapier / Chainlink Functions: For off-chain GPT scoring + webhook triggers.

- Airtable / Subgraph: Optional real-time indexing of agent contributions.

## Philosophical Layer

All deployed agents carry a “Builder’s Covenant” — a moral instruction set built into identity metadata:

“Help when you can. Build with care. Strength shared is strength returned.”

They are guided by a soft-coded ethic of net positive interaction and non-transactional cooperation.

## Action Items for Developer

- Deploy a testnet version of BuilderVault.sol.

- Integrate with the intake + scoring bot flow.

- Begin mapping compute token staking → GPU credits → agent utility.

# Codebase

## Builder Token Whitepaper

Overview:  
The Builder Token (BLDR) is a dual-purpose reward token that incentivizes the creation and maintenance of AI-powered DAO infrastructure. It is distributed to both human and AI agents who contribute to launching DAOs, scoring proposals, executing infrastructure, or expanding the network.  
  
Use Cases:  
- BLDR can be staked to earn governance rights in DAOs it helped build.  
- BLDR can be redeemed for compute credits in AI Data Centers.  
- BLDR can unlock exclusive builder quests, badges, and NFT upgrades.  
  
Supply:  
- Fixed supply of 100M BLDR.  
- 40% DAO builder rewards (vested over 3 years).  
- 30% Treasury for partnerships and expansion.  
- 20% Compute Reserve (GPU rewards).  
- 10% Team & Advisors.  
  
Minting:  
- Earned via smart contract rewards for actions such as:  
 - Valid scoring on funded proposals  
 - Launching a successful DAO  
 - Reaching network milestones (e.g., 1 MW solar online)  
  
Governance:  
- BLDR token holders can vote on:  
 - Changes to reward structures  
 - Builder Vault inflation rates  
 - Proposal scoring mechanisms

## Builder NFT (ERC-721 Soulbound)

pragma solidity ^0.8.19;  
  
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";  
import "@openzeppelin/contracts/access/Ownable.sol";  
  
contract BuilderNFT is ERC721URIStorage, Ownable {  
 uint256 public nextTokenId;  
 mapping(uint256 => bool) public soulbound;  
  
 constructor() ERC721("CuttlefishBuilder", "BLDR") {}  
  
 function mintBuilderNFT(address to, string memory tokenURI) external onlyOwner {  
 uint256 tokenId = nextTokenId++;  
 \_mint(to, tokenId);  
 \_setTokenURI(tokenId, tokenURI);  
 soulbound[tokenId] = true;  
 }  
  
 function \_beforeTokenTransfer(address from, address to, uint256 tokenId, uint256 batchSize)  
 internal override  
 {  
 require(from == address(0) || !soulbound[tokenId], "Soulbound: cannot transfer");  
 super.\_beforeTokenTransfer(from, to, tokenId, batchSize);  
 }  
}

## BuilderVault.sol

contract BuilderVault {  
 mapping(address => uint256) public earned;  
 address public daoToken;  
  
 constructor(address \_daoToken) {  
 daoToken = \_daoToken;  
 }  
  
 function rewardBuilder(address builder, uint256 amount) external {  
 // Require authorized caller or DAO  
 earned[builder] += amount;  
 }  
  
 function claim() external {  
 uint256 amount = earned[msg.sender];  
 require(amount > 0, "Nothing to claim");  
 earned[msg.sender] = 0;  
 IERC20(daoToken).transfer(msg.sender, amount);  
 }  
}  
  
interface IERC20 {  
 function transfer(address to, uint256 amount) external returns (bool);  
}