ChatGPT

Migrating Ethereum DApps to Polkadot – Technical Roadmap & Strategy

Overview: We will migrate three Ethereum projects – **Worboo**, **VaultCraft**, and **WhisperFi** – from a Solidity + Hardhat stack to the Polkadot ecosystem. The plan is to start with the simplest (Worboo) and progressively tackle more complex features (VaultCraft's DeFi vaults, then WhisperFi's ZK-powered privacy). This phased approach allows us to incrementally adapt to Polkadot's technology while **leveraging Polkadot's new EVM compatibility (REVM)** and developer tooling support. Polkadot's 2025 upgrades emphasize running **unchanged Solidity code with familiar tools** [1] [2], so we aim to reuse as much of our existing code and workflow as possible, while integrating Polkadot-specific features for bonus hackathon points.

Stage 1: Porting Worboo to Polkadot EVM

Goal: Deploy the **Worboo** project on a Polkadot-based platform with Ethereum Virtual Machine support. This will validate our basic contracts and front-end on Polkadot with minimal modifications.

- **Choose a Deployment Environment:** Start with an EVM-compatible Polkadot parachain or testnet. The quickest path is **Moonbeam** (a Polkadot parachain with full Ethereum compatibility) or the new **Polkadot Contracts Hub (Asset Hub)** EVM environment if available. Polkadot's upcoming **Asset Hub** supports Solidity contracts via a **pallet-REVM** (Rust EVM) backend [3] [4], meaning we can deploy unmodified Solidity code on Polkadot. For initial testing, Moonbeam's stable EVM may be easiest; later we can try Polkadot's native hub when it's live on Kusama/ Polkadot.

- **Migration Steps:**

- **Configure Tools for Polkadot:** Point Hardhat (or Foundry) to the Moonbeam RPC or Polkadot EVM RPC endpoint. Frontier/REVM provides an Ethereum JSON-RPC interface, so our Hardhat scripts and Metamask should work out-of-the-box [2]. For example, we'll add a network config for Moonbeam (or the Westend Asset Hub testnet) with chain ID, RPC URL, and deploy as if to Ethereum.
- **Deploy Smart Contracts:** Deploy Worboo's Solidity contracts to the Polkadot EVM. If Worboo is an ERC-20/721 or simple DApp, no code changes are needed – Polkadot's EVM pallet aims for **full Ethereum opcodes compatibility** [2]. We will verify that Polkadot's EVM has all required precompiles (e.g., for any cryptography or hashing used) and adjust gas limits if needed (Polkadot may have different block gas limits or pricing).
- **Frontend Integration:** Update Worboo's front-end dApp to connect to Polkadot. This likely means switching the network in web3 providers to the chosen Polkadot chain and ensuring Polkadot's **account format vs Ethereum addresses** is handled. (The Polkadot EVM will map 160-bit Ethereum addresses to 256-bit Substrate accounts behind the scenes [5] [6], so tools like Metamask can be used directly with an Ethereum-style address.)

- **Testing:** Thoroughly test all Worboo functions on the Polkadot network (using a local node or testnet). Thanks to Polkadot's focus on **Hardhat and Foundry integration** [7], we can use our existing Hardhat tests and even Forge/Anvil for simulations. We’ ll confirm that transactions and events behave as expected and that the UI works with Polkadot’ s chain RPC.

- **Key Challenges & Solutions:**

- Environment Setup: Learning to run a local **Polkadot node with EVM** support. We can use a Moonbeam standalone node or the official Polkadot Contracts Hub test node. Polkadot provides an Anvil-compatible test node for Ethereum tooling [7], which simplifies local debugging.
- Differences in Accounts: Polkadot uses SS58 addresses normally, but in the EVM context this is abstracted. We must ensure our Ethereum private keys can control the corresponding Polkadot EVM accounts (Polkadot’ s EVM pallet implements a two-way mapping between Ethereum 0x addresses and Polkadot accounts [5]).
- Gas and Fees: Polkadot’ s runtime uses weight-based fees, but the EVM pallet maps Ethereum gas to Substrate weights so that Ethereum wallets behave normally [3] [8]. We should verify Worboo’ s transactions fit within these gas limits and that the fee paid in the Polkadot network’ s token (e.g. GLMR on Moonbeam or DOT on Polkadot Asset Hub) is acceptable.
- Randomness or Oracles: If Worboo relies on Ethereum-specific infrastructure (e.g., Chainlink or `block.timestamp` behavior), we may need Polkadot equivalents. For basic DApps, this is likely minimal.

Outcome: By the end of Stage 1, Worboo should be running on Polkadot’ s EVM platform, demonstrating that a simple Solidity project can **run unchanged on Polkadot** [9]. This gives the team familiarity with Polkadot’ s Ethereum compatibility and toolchains, setting the stage for more complex migrations.

Stage 2: Migrating VaultCraft – DeFi Vaults on Polkadot

Goal: Adapt **VaultCraft**, originally an Ethereum DeFi vault protocol (likely using yield strategies and ERC-4626 vault standards), to Polkadot. This stage introduces management of assets and potentially cross-chain interactions.

- **Deployment Environment Options:** We have two main paths:
- **Use an EVM Parachain** (Moonbeam/Astar) – Continue leveraging Ethereum compatibility. This allows reusing VaultCraft’ s Solidity code and perhaps integrating with any EVM DeFi protocols available on that chain (e.g., if Astar/Moonbeam have DEXes or yield sources). It’ s the straightforward approach, treating Polkadot just like another EVM chain.
- **Build a Custom Parachain** – If VaultCraft needs custom runtime features (for example, tighter integration with Polkadot’ s multi-asset system or on-chain scheduling), we could use the **Polkadot SDK** (Substrate) to create our own parachain that includes the Frontier EVM pallet. With Frontier, our chain would natively run Ethereum contracts and also allow custom pallets for specialized logic [10]. This is more complex but could fit the hackathon “Build a chain” track if we choose it, and it lets us optimize VaultCraft’ s functionality at the runtime level.

- **PolkaVM (Pallet-Revive)** – As Polkadot’ s **native PolkaVM** goes live, we could deploy VaultCraft on the Polkadot Asset Hub using the PolkaVM execution engine instead of a separate parachain. PolkaVM runs Solidity via a RISC-V JIT, offering better performance and potentially lower fees [11] [1]. The advantage is we still write Solidity, but behind the scenes Polkadot uses a new VM. We might not need to change VaultCraft’ s code for this; simply compile and deploy through Polkadot’ s provided pipeline (solc → Yul → RISC-V via the Revive compiler [12]). For hackathon purposes, demonstrating VaultCraft on PolkaVM (if timing allows on testnet) would earn innovation points.

- **Adapting to Polkadot Features:** VaultCraft likely deals with depositing tokens and earning yield. On Polkadot, we should leverage the **multi-chain asset ecosystem**:

- **Asset Integration:** Identify Polkadot equivalents for assets used in VaultCraft. For example, if VaultCraft vaults accept USDC/ETH on Ethereum, on Polkadot we might use XC-20 assets (cross-chain assets) or native assets like DOT or aUSD. The **Asset Hub** will allow accessing assets like USDT/USDC via precompiled ERC-20 contracts [13] [6] – meaning our EVM code could potentially handle DOT or USDT as ERC-20 tokens through Polkadot-provided interfaces. We should plan VaultCraft vaults around available yield-bearing assets on Polkadot (e.g., LDOT – liquid DOT, or interest-bearing aUSD, etc.).
- **Yield Sources:** Ethereum vaults earn yield by interacting with protocols (e.g., Yearn vault investing in Compound/Aave). On Polkadot, if similar lending or staking protocols exist on the chosen chain, we can redirect VaultCraft's strategy contracts to those. For instance, on Moonbeam we might integrate with **StellaSwap or Moonwell** for yield, or use Polkadot's staking (via Liquid Staking tokens) as a source of yield. If no direct analog exists, a simpler approach is to simulate yield (e.g., a mock interest mechanism) or leverage cross-chain messaging: Polkadot's **XCM** could allow a vault on one parachain to deposit assets into a yield protocol on another parachain. For hackathon scope, we might limit complexity by focusing on one chain's ecosystem or using a placeholder yield model.

- **Cross-Chain Interoperability (Future):** A stretch goal is to show VaultCraft vaults that accept deposits on one chain and deploy across multiple chains for best yield. Polkadot's XCM SDK and upcoming cross-chain libraries could enable this. For now, we note it as future work but can highlight architecture in our presentation to impress judges.

- **Technical Challenges for VaultCraft:**

- Complex Contract Logic: VaultCraft's contracts (ERC-4626 vaults, strategies, fee mechanisms) need to run within Polkadot's EVM gas limits and possibly with different block times. We'll use Polkadot's **Anvil-compatible local node** to run our test suite [7] and ensure all logic holds.
- Asset Interfaces: We must ensure the ERC-20 interface on Polkadot assets is standard. If using Asset Hub assets, we might call into a precompile (provided by Polkadot's runtime) that wraps DOT or XC-20 as an ERC-20. Polkadot's EVM compatibility is adding support for **18-decimal DOT** for ease of integration [14], which simplifies using DOT in Ethereum contracts (no need to handle 10 decimal to 18 decimal conversion).
- Security and Differences: Polkadot's block time (6 seconds) and finality (GRANDPA) differ from Ethereum. Time-based strategies (if any) in VaultCraft should be reviewed. Also, Polkadot's randomness or price feeds might require using an oracle (there's no Chainlink on Polkadot mainnet yet, but parachains like Moonbeam have oracle feeds). We might rely on existing price oracles on Moonbeam for any strategy logic.
- Front-end Updates: VaultCraft's dApp front-end must be updated to display Polkadot network specifics (e.g., DOT balances, Polkadot wallet connectivity via Talisman or Polkadot.js if not purely MetaMask). Using **Polkadot JS API** or Ethers.js should be decided based on whether we stick to Ethereum API or need substrate queries. Since we deploy via EVM, Ether.js (via the provided RPC) suffices for most interactions.

Migration Path: We plan to **first deploy VaultCraft on Moonbeam's EVM** as a proof of concept (quickest way to get it running). Then, time permitting, experiment with **Polkadot's Asset Hub** (using REVM/PolkaVM) by deploying the same contracts on the Westend Asset Hub testnet. This two-step deployment shows both a near-term solution and a forward-looking approach with Polkadot's latest tech. We will resolve any contract issues on Moonbeam, then tackle PolkaVM's slight differences (e.g., ensuring our Solidity compiler output works with the Revive toolchain). Polkadot’s documentation indicates that even older Solidity versions are supported in PolkaVM due to the Yul compilation approach [4] [12], so VaultCraft’s code (assuming Solidity ^0.8.x) should be compatible.

Outcome: By end of Stage 2, VaultCraft will be operational on a Polkadot parachain, managing Polkadot-based assets in vaults. We’ll have demonstrated using **Ethereum DeFi logic on Polkadot** while considering Polkadot’s multi-asset and cross-chain strengths. This positions us well for hackathon judging, as we address both compatibility and innovation (possibly showcasing an asset-bridging or multi-chain feature).

Stage 3: Porting WhisperFi – ZK and Privacy on Polkadot

Goal: Migrate **WhisperFi**, which includes a zero-knowledge (ZK) module and possibly account abstraction, to Polkadot. This is the most complex stage, involving privacy tech and potentially AI integration (as hinted by WhisperFi’s description as AI-driven DeFi). We aim to preserve its core functionality (private transactions or AI-managed strategies) using Polkadot-compatible ZK tools.

- **Deploy on EVM vs Custom Approach:** Given WhisperFi’s complexity, we have a decision to make:
- **EVM Parachain Deployment:** Deploy WhisperFi’s Solidity contracts on Moonbeam/PolkaVM similar to previous stages. If WhisperFi uses ZK SNARKs (e.g., a circuit for anonymity like TornadoCash or some proof verification), we can likely use Ethereum-equivalent approaches. Polkadot EVM supports the same BN254 elliptic curve precompiles (`ALT_BN128`) as Ethereum, so verifying zkSNARK proofs in a Solidity contract is possible. For example, if WhisperFi uses a Circom-generated verifier contract, it should run on Polkadot’s EVM without issues. This approach is straightforward: use **off-chain SNARK proof generation** (perhaps via the existing zk module code) and on-chain verification with the Solidity verifier contract.
- **Native ZK Integration (Substrate):** For a more ambitious path, build a Substrate pallet or off-chain worker to handle ZK proof logic. Polkadot’s design allows heavy computation in **off-chain workers** (background tasks run by validators) which could generate or verify proofs outside the normal transaction flow [15]. For instance, we could create a pallet that triggers an off-chain worker to run a Zero-Knowledge proof (using, say, the **Risc0 ZKVM** or other ZK-WASM frameworks) and then have a lightweight proof verified on-chain. This could be useful if WhisperFi’s use-case is complex (e.g., proving ML computations or cross-chain data correctness). Given hackathon time constraints, fully implementing this is difficult, but we can outline this approach in our Future Development plans.

- **Use a ZK-Friendly Chain or Tool:** Leverage Polkadot ecosystem projects focused on ZK. For privacy transactions, **Manta Network** or **Aztec (if bridged)** could provide inspiration, but integrating them might be out of scope. Alternatively, tools like **zkSync or Polygon CDK (ZK)** are Ethereum-centric; on Polkadot, an equivalent might be writing the logic in **ink! and using Arkworks** (Rust ZK libraries) for proofs on a contract chain, or using upcoming **ZKcoprocessor services**.

- **Account Abstraction (AA):** WhisperFi mentions account abstraction (possibly to let users interact via smart contract wallets or meta-transactions). Ethereum’s AA (ERC-4337) won’t natively exist on Polkadot EVM. To emulate AA on Polkadot:

- On an EVM chain, we could deploy the standard ERC-4337 contracts (EntryPoint, etc.), but without the bundler infrastructure and Paymasters it may not be fully functional. Instead, we might

simulate AA by using a simple relayer: allow users to sign actions and have a relayer submit them (the relayer could be a backend we run during the hackathon).
- On a custom chain, Polkadot allows true account abstraction at the runtime level (we could customize how transactions are validated, e.g., accept zkSNARK proofs as signatures). This is advanced and not feasible to implement fully during the hackathon, but we will mention that Polkadot's flexibility allows such designs (and indeed Polkadot is exploring Account Abstraction in its roadmap).

- **Plan:** Implement basic meta-transaction capability for WhisperFi on the EVM platform (so users sign messages off-chain and our dApp or a server submits actual transactions). This provides a smoother UX (no need for users to pay gas directly, aligning with AA goals) and is achievable with existing libraries.

- **ZK Tooling and Integration:** The core of WhisperFi's migration is handling its ZK component:

- **Risc0 ZKVM:** A promising route is to use **Risc0** for any complex computations that need to be proven. Risc0 lets you write general-purpose code (in Rust or C++) that executes in a RISC-V VM and produces a STARK proof of its execution. This aligns with Polkadot's RISC-V direction and could allow us to prove off-chain computations (e.g., AI model decisions or private data transformations) and verify a concise proof on-chain. For example, if WhisperFi's AI suggests trades, a Risc0 program could prove "Given input X, the recommended trade was Y" without revealing sensitive data. We could then verify that proof on-chain by either:
    - Using a Solidity verifier for Risc0's proofs (Risc0's STARK can be recursively verified or converted to a SNARK [16]). This might require integrating Risc0's proof system with a pairing-based proof for on-chain verification – an advanced step likely beyond hackathon scope, but worth mentioning.
    - Or, verify incrementally via a custom pallet that understands Risc0 proofs (if we built our own chain). A Web3 Foundation grant project hints at using Risc0 for off-chain proofs in Substrate [15], confirming feasibility.
- **ZK-WASM / ZK-CoProcessors:** We'll research emerging **"ZK copilot"** solutions – essentially services or frameworks that help blockchains offload and verify computations. For instance, **Brevis Network** provides a ZK data coprocessor (marketed as a "computational copilot" for smart contracts) which can prove cross-chain data or off-chain computation to on-chain contracts [17]. Similarly, **Lagrange** is building a decentralized ZK proof network (also referred to as a "ZK copilot") to supply verifiable computation on demand [18]. In a forward-looking sense, WhisperFi could integrate such a service: e.g., use Brevis to fetch and prove external data (like user credit scores or off-chain analytics) into Polkadot, enhancing its AI-driven decisions with trustless data feeds. While we likely can't fully integrate this during the hackathon, we will emphasize this path for future development.
- **ZK-SNARK Circuits:** If WhisperFi already has circuits (perhaps written in Circom or ZoKrates), we will port those. Circom circuits can be compiled to work with the BN254 curve used on Ethereum, so verifying them on Polkadot EVM is fine. We just need to set up the circom toolkit in our environment (circom, snarkjs, etc.) and generate new trusted setups if needed for the Polkadot deployment (or reuse the existing ones if the circuit didn't change). We'll double-check Polkadot's EVM precompile support for pairings – Frontier/REVM should support the same set of precompiles including MiMC, SHA, BN128 pairings, etc., so a TornadoCash-like verification should succeed.

- **Performance:** ZK proofs can be heavy. We should be mindful of Polkadot's block time and weight limits. Verifying a Groth16 proof on-chain is fast (a few ms) but generating it might be slow (seconds). During hackathon demo, we can pre-generate proofs for certain scenarios to save time. If using Risc0, proof generation is slower (possibly minutes) so likely offloaded to a server or just conceptual. It’s acceptable to demonstrate one proof verification rather than live generation in the judging.

- **UI/UX Considerations:** WhisperFi being a user-facing DeFi app with AI, we want to polish the front-end to meet Polkadot’s hackathon guidelines:

- Implement Polkadot’s **design guidelines** for the UI [19] – adopting Polkadot branding where appropriate (colors, fonts) to show we followed the requirements.
- Ensure the app connects with popular Polkadot wallets (if using EVM, MetaMask on Moonbeam; if on Polkadot Asset Hub, maybe Talisman or a special provider). Possibly use **Polkadot.js** extension if any substrate queries are needed.
- Highlight the **privacy features** in the UI, showing how user data or strategies are protected by ZK proofs (even if under the hood we did an off-chain generation).
- Provide a smooth onboarding since hackathon judges will actually run the app on the Polkadot hub environment (per rules, all projects will be run on the Polkadot Hub platform) [20].

Outcome: By the end of Stage 3, we aim to have WhisperFi’s core functions deployed on Polkadot, at least in a demonstrative capacity. This includes any smart contracts (e.g., for private transactions or managing AI-driven vaults) running on Polkadot EVM/PVM, and a working demonstration of how a zero-knowledge proof submitted by a user is verified on-chain to enable a private or AI-informed action. We will document the limitations (e.g., if some privacy features are simulated due to time) and propose how Polkadot’s evolving tech (PolkaVM for faster ZK, off-chain workers, etc.) can fully realize WhisperFi on Polkadot in the near future.

Recommended Toolchain & Compatibility

Migrating to Polkadot doesn’t mean abandoning our Ethereum developer tools. On the contrary, Polkadot’s ecosystem in 2025 is explicitly **designed to accommodate Ethereum tooling** [1] [21]. Here’s our toolchain strategy:

- **Solidity Compilers:** We will continue writing in Solidity. For EVM deployments, we use the standard `solc` (via Hardhat or Foundry). If we target PolkaVM, Polkadot provides a modified pipeline where `solc` produces Yul IR and then the **Revive** compiler converts Yul to RISC-V bytecode [12]. This is transparent to us when using Polkadot’s Remix or RPC tools. We should test compilation both via Hardhat (for EVM bytecode) and via Polkadot’s Remix (for PolkaVM) to ensure compatibility.

- **Hardhat and Foundry:** These remain our primary development and testing frameworks. We’ll use **Hardhat** for deployment scripts, network configuration, and plugin support (e.g., verifying contracts on Moonbeam’s block explorer). **Foundry (Forge + Anvil)** will be extremely useful for fast testing and for its cheat code capabilities in a local environment. Polkadot’s teams are making Foundry’s Anvil work with their stack (even enabling forking of live Polkadot chains in testing) [7]. For example, we can use Anvil to fork the Moonbeam devnet or the Westend Asset Hub, and run our Solidity tests against that state. This helps catch any Polkadot-specific quirks early. We’ll maintain our Hardhat tests but likely port many to Foundry for speed.

- Hardhat Compatibility: When using Polkadot’s Ethereum API (through Frontier or the RPC proxy), Hardhat tasks like `hardhat deploy` and `hardhat test` should function normally. We might incorporate the [polkadot-hardhat-plugin]() if available, which ensures Hardhat knows about the chain’s block time and gas.

- Foundry Benefits: Forge tests can directly call our contracts in a local Anvil instance configured for Polkadot. The hackathon encourages this, as they are developing an "Anvil-compatible node" for Polkadot EVM [7]. We' ll use this to simulate scenarios (especially for VaultCraft and WhisperFi) quickly, including fuzz testing vault math or verifying that a zkSNARK verification doesn' t exceed gas limits.

- **Solang & Ink! (Optional):** As an exploratory step, we might try the **Solang compiler** to compile Solidity to WASM, enabling deployment on an ink! smart contract chain (like Astar' s WASM or a local substrate node with Contracts pallet). This isn' t strictly necessary – since Polkadot now offers a direct Solidity path via EVM/PolkaVM – but it could be educational. For instance, we could compile a small contract from Worboo using Solang and deploy it on a Wasm contract chain to compare performance or security. If we identify any piece of our projects that might benefit from Rust' s safety or integration, we could rewrite it in **ink!** and use it alongside EVM contracts (Astar supports hybrid EVM and Wasm contracts). However, given time constraints, this is lower priority. Our focus will remain on the **EVM-compatible route** which maximizes reuse of code.

- **Polkadot SDK & Substrate:** If we pursue the custom parachain path (especially for VaultCraft or to integrate WhisperFi' s features at the chain level), we' ll use the Polkadot SDK (Substrate framework). This involves:

- Starting from a Substrate node template or the official Polkadot-Parachain templates.
- Adding the **Frontier pallets** (pallet-evm, pallet-ethereum, pallet-base-fee, etc.) to get an EVM execution environment [22]. We would configure it to use the **pallet-revm (Revive)** backend instead of the legacy Sputnik engine, to align with the latest Polkadot improvements.
- Registering any additional pallets: e.g., a custom Vault pallet or a ZK verification pallet if we want on-chain SNARK verification outside the EVM (Substrate has an `alt_bn128` crate we could utilize for native Rust verification, or even integrate Arkworks for different curves).
- Using **Cargo + Rust** toolchain for runtime development, and **Polkadot Launcher** or **Zombienet** for testing the custom chain.

- This is quite advanced for a hackathon; thus, our default plan is **not** to build a custom chain unless we find a compelling reason and have sufficient time/team size. We will prepare the environment (install Rust toolchain, compile the node) in parallel, so that if needed, we can quickly pivot to building specific runtime features (e.g., a specialized pallet for WhisperFi).

- **Testing & Deployment Tools:**

- We' ll use **Polkadot JS Apps** and/or **Block Explorers** (like Moonscan for Moonbeam, or Subscan) to monitor our contracts on Polkadot. This ensures we can inspect events, storage, and ensure things are working on-chain.
- **Chopsticks or Zombienet:** These Polkadot testing tools allow spinning up a local multi-chain environment or even forking an existing chain state. If VaultCraft needed to test XCM calls, we could use Zombienet to simulate two chains (one hosting the vault, another providing a yield source). Chopsticks can fork a live chain state similar to Anvil' s forking but for Substrate. These are bonus tools if needed for complex integration testing.
- **CI/CD:** We might set up a simple GitHub Actions pipeline to run our Forge/Hardhat tests on each commit, ensuring we don' t break anything during the migration.

In summary, our toolchain is a **blend of Ethereum favorites (Hardhat, Foundry, Solidity)** and **Polkadot-specific tools (Substrate SDK, Polkadot.js)**, chosen to maximize productivity and compatibility. Polkadot' s Frontier and upcoming PolkaVM explicitly allow Ethereum devs to "bring their own tools" with minimal friction [9], so we leverage that fully. This combination will help us iterate quickly during the hackathon while satisfying the technical requirements.

Aligning with Hackathon Goals and Maximizing Scores

To succeed in the Polkadot 2025 hackathon, we must not only port our projects but also showcase innovation and compliance with the event's rules/directions. Here are our strategies to score high:

- **Meet All Submission Requirements:** We will ensure our project is fully open-source and follows the prescribed format. This includes using the provided **UI template** (forking the Polkadot Hackathon dotUI kit) and following Polkadot's branding in our web app [19]. All team members will verify their identities on the OpenGuild Discord as required [23]. We'll maintain a clear commit history on GitHub during the hackathon to show our development progress (which is part of judging) [24].

- **Use Polkadot Technology Emphatically:** The hackathon specifically encourages leveraging **REVM (Rust EVM) and Ethereum-tooling compatibility** [4] – essentially rewarding projects that make Polkadot accessible to Ethereum devs. Our migration itself is a testament to this goal, and we will emphasize that:

- Worboo was moved with zero Solidity changes, highlighting Polkadot's compatibility.
- VaultCraft uses Polkadot's multi-chain assets via XCM, demonstrating unique Polkadot functionality on top of EVM.

- WhisperFi uses cutting-edge Polkadot ZK support (e.g., PolkaVM's efficiency or off-chain workers), aligning with Polkadot's forward-looking features. By doing so, we implicitly support Polkadot's narrative of **"bring your Ethereum dApp and get more features"**. We will cite how our use of PolkaVM improved performance or how using XCM unlocked cross-chain capability, to show Polkadot not only preserved but augmented our dApps.

- **Innovation & Theme Fit:** The hackathon tracks likely include categories like Web3 Applications, Infrastructure/Chains, Cross-Chain Innovations, etc. Our plan touches multiple:

- By porting three projects, we're essentially creating a **suite of Web3 dApps** on Polkadot, which fits an "Apps that Change the World" narrative (e.g., WhisperFi could be pitched as a privacy-preserving DeFi AI app – very on-trend).
- If we do the custom chain route or deeply integrate PolkaVM, we enter the **"Build a chain with Polkadot"** track, showcasing a parachain that hosts our products. Even if we don't go fully custom, we can frame our Polkadot Asset Hub deployment as building on Polkadot's core chain.
- We integrate **interoperability** by design. Polkadot's USP is cross-chain, so we plan for VaultCraft's cross-chain vault idea or WhisperFi pulling data from multiple chains with ZK proofs. Even partial implementation or a demo of moving an asset via XCM (for example, vault depositing DOT from Relay Chain to an Asset Hub account) will score innovation points.

- **ZK and Security:** WhisperFi's ZK emphasis hits the privacy track and technical complexity, which judges often score well if explained clearly. We will highlight how Polkadot's features (RISC-V PolkaVM, off-chain workers) make it easier or more scalable to do ZK compared to vanilla Ethereum – thus aligning with hackathon themes of next-gen blockchain tech.

- **Polish and Presentation:** We will produce a clear **architecture diagram** showing our migration path – e.g., Ethereum architecture vs Polkadot architecture for each project – to visualize the journey. This will be included in our documentation or slides. Such diagrams can illustrate how Frontier/REVM sits in the stack, how XCM connects chains, or how Risc0 off-chain proofs feed into on-chain verification. A polished presentation that tells the story of "Ethereum apps enhanced by Polkadot" will resonate with judges.

- We'll also document challenges and how we overcame them (as required in submission) [25], which gives the impression of deep problem-solving – always a plus in hackathons.
- If possible, we will deploy a **live demo on Polkadot's testnet** and provide a link. Since the hackathon environment is contracts.polkadot.io, we assume a testnet or local setup will be used. We'll script an easy way for judges to run our projects (perhaps a simple script or a hosted demo frontend connected to a test network).
- **Bonus – Community & Common Good:** Polkadot hackathons often appreciate if projects consider **open good** or future ecosystem contribution. By open-sourcing our migrated code under a permissive license and perhaps writing a short guide "Ethereum to Polkadot migration tips" in our repo, we position our project as helpful to others (common-good spirit). Additionally, since our code will be forked into the OpenGuild Labs org for submission [26], we want it to be in a state that others can learn from or build on.

In essence, our approach to maximize hackathon success is: **demonstrate technical excellence in bridging Ethereum and Polkadot, highlight the new Polkadot features (REVM, PolkaVM, XCM, etc.), and present a compelling, polished product story.** By ticking all the boxes – compliance, innovation, usability, and documentation – we aim to score top marks.

Future-Proofing WhisperFi: ZK and Beyond on Polkadot

Migrating WhisperFi is not the end – it's a beginning. Here we outline forward-looking ideas to fully harness Polkadot's evolving tech stack for a zero-knowledge DeFi project:

- **Adopt Risc0 for Off-chain ZK Compute:** As discussed, integrating Risc0 can greatly enhance WhisperFi. In the future, we envision WhisperFi running critical computations (like simulation of investment strategies or private data analysis) inside a Risc0 zkVM. Polkadot could verify these via on-chain SNARK proofs. Notably, Polkadot's move to RISC-V (PolkaVM) hints at eventual **ZK-friendliness of the runtime** – perhaps one day the PolkaVM will itself produce proofs of contract execution. We'll keep an eye on that, but meanwhile Risc0 provides a prove once, verify anywhere model. We would work on optimizing proof generation (perhaps using GPU acceleration, as projects like Tachyon are doing for zkEVM proving) to eventually get near-real-time proofs. This aligns with Polkadot's scalability ethos and would make WhisperFi incredibly robust (users could trust AI decisions or private computations without trusting a server).
- **Leverage "ZK Copilot" Networks:** The concept of a **ZK coprocessor (ZK Copilot)** is gaining traction [27]. In practice, this means WhisperFi could outsource heavy tasks to a decentralized service that returns a succinct proof. For example, **Brevis** could allow WhisperFi to query historical data from Ethereum or off-chain APIs in a trustless manner – the coprocessor returns a proof that "this data is genuine". WhisperFi can then use that data (e.g., an AI algorithm's input) without needing oracles or manual verification. Similarly, **Lagrange's network** might let us tap into a distributed network of provers for AI model inference (imagine proving that an AI model's output meets certain criteria, using their GPU-accelerated ZK cloud). As these services mature (some are funded in late 2024), WhisperFi should integrate them to become an **AI-Enhanced, cross-chain**

DeFi platform with built-in trust guarantees. It’s exactly the kind of cutting-edge composition Polkadot enables – connecting to off-chain and cross-chain systems with proofs instead of trust.

- **ZK-WASM and WASM Contract Support:** Polkadot’s support for WebAssembly smart contracts (via ink! and the Contracts pallet) offers another route for ZK. Projects like **zkSync’s VM** or **ZKWASM** frameworks let you write high-level code that is proven in zero-knowledge. In Polkadot, an ink! smart contract could potentially verify a ZK-WASM proof or even run a ZK circuit if light enough. In the future, we might port parts of WhisperFi to an ink! contract to take advantage of Rust’s strong cryptography libraries (like Arkworks for Groth16, Nova for recursive proofs, etc.). For instance, a Rust contract could verify a Plonk proof more efficiently than a Solidity contract on EVM. There’s also ongoing research on making **WASM itself a proving target** – i.e., proving the execution of a WASM contract off-chain. If Polkadot supports that, we could have WhisperFi’s contract logic (written in Rust or AssemblyScript) run as a ZK computation, giving ultimate privacy (the chain only sees proofs, not actual user data).

- **Account Abstraction on Polkadot:** Polkadot is exploring native account abstraction (AA) in its design (with concepts like “Account Polymorphism” in substrate). We anticipate that by the time Polkadot 2.0 comes (perhaps after this hackathon), developers might be able to implement custom signature verification or payment logic at the runtime level. WhisperFi could evolve to utilize this: for example, allowing users to interact using **smart contract wallets** on Polkadot that can be sponsored by relayers or tied to identity registries. Also, because Polkadot uses a different transaction format, we could incorporate AA features like **session keys, multi-sig, social recovery** directly via substrate pallets (e.g., the Social Recovery pallet, etc.). In short, as Polkadot’s base layer adds flexibility in how transactions are authorized, we will integrate those features to make WhisperFi user-friendly and secure (no more private key management worries for end-users, if we can use AA to allow biometrics or multi-factor auth, for instance).

- **Continuous Compatibility with Ethereum:** Since WhisperFi originated in Ethereum, we won’t abandon Ethereum entirely. We plan to maintain interoperability – e.g., if WhisperFi uses an L2 or sidechain on Ethereum for some ZK proofs, we could bridge results to Polkadot. Polkadot’s upcoming **Snowbridge** (Ethereum-Polkadot bridge using zk proofs for trustlessness [28]) might allow us to trustlessly use Ethereum data or even move WhisperFi’s assets between Ethereum and Polkadot. Keeping one foot in Ethereum’s ZK ecosystem (which is very active) while deploying on Polkadot’s infrastructure could give us the best of both worlds. We’ll monitor projects like Axiom, Scroll, zkEVMs, etc., and adopt any tech that can plug into Polkadot (perhaps via an XCM interface or a light client pallet).

Conclusion: By following this migration roadmap, we start with “low-hanging fruit” (simple contract deployment on Polkadot) and progressively integrate more of Polkadot’s advanced features (cross-chain, ZK, custom runtimes). Each project’s migration builds on the previous, honing our skills and confidence. In the end, we plan to have a cohesive suite of applications running on Polkadot, taking full advantage of Polkadot’s EVM compatibility and beyond. This positions us strongly for the hackathon – we demonstrate not just a port, but an evolution of our projects aligned with Polkadot’s vision for 2025. By addressing potential challenges and having a forward-looking plan (especially for the ZK-heavy WhisperFi), we also ensure that our Polkadot-based products remain **cutting-edge, scalable, and secure** well past the hackathon.

Throughout the journey, our guiding principle is to **embrace Polkadot’s technology without losing the Ethereum developer experience**. Thanks to initiatives like Frontier and PolkaVM, this is feasible – Solidity and VMs on Polkadot operate seamlessly, letting us focus on higher-level improvements. Ultimately, this migration could serve as a showcase for how Ethereum projects can **transform into**

multi-chain Polkadot powerhouses with relatively low friction [9] [2], which is exactly what the hackathon aims to encourage.

Sources:

- Polkadot Forum – Smart Contracts on Polkadot Hub: Progress Update (Aug 2025) [3] [4] [7] [5]
- Medium (OneBlock+) – Polkadot embracing RISC-V & PolkaVM (Apr 2025) [1] [9]
- Frontier Documentation – Ethereum Compatibility Layer for Substrate [2] [22]
- Polkadot Hackathon 2025 Repo – Rules and Guidelines [19] [24]
- Polkadot Forum – Hyperfridge: ZK for TradFi (using Risc0) [15]
- Binance/Media – Brevis zkCoproc intro (ZK Copilot concept) [17]

---

[1] [9] Beyond Coincidence: Why Are Polkadot and Ethereum Embracing RISC-V? | by OneBlock+ | Medium
https://medium.com/@OneBlockplus/beyond-coincidence-why-are-polkadot-and-ethereum-embracing-risc-v-1a15f4a0094f

[2] [10] [22] Frontier
https://polkadot-evm.github.io/frontier/

[3] [4] [5] [6] [7] [8] [11] [13] [14] [21] Smart Contracts on Polkadot Hub: Progress Update - Ecosystem - Polkadot Forum
https://forum.polkadot.network/t/smart-contracts-on-polkadot-hub-progress-update/14596

[12] In-Depth Analysis of PolkaVM: A Perfect Path to Understanding Polkadot 2.0 | by OneBlock+ | Medium
https://medium.com/@OneBlockplus/in-depth-analysis-of-polkavm-a-perfect-path-to-understanding-polkadot-2-0-6ac347a296ba

[15] [16] Hyperfridge - ZK-Web-Proofs for tradional financial backends (TradFi) - Ecosystem - Polkadot Forum
https://forum.polkadot.network/t/hyperfridge-zk-web-proofs-for-tradional-financial-backends-tradfi/8637

[17] TO฿EXOLA on X: "Yesterday, I made a bold pivot to @brevis_zk ...
https://x.com/exola_g/status/1978001722510299512

[18] Latest #Web3Infra News, Opinions and Feed Today | Binance Square
https://www.binance.com/en-IN/square/hashtag/Web3Infra

[19] [20] [23] [24] [25] [26] GitHub - openguild-labs/polkadot-hackathon-2025: Repository to host all submission for the Polkadot hackathon 2025
https://github.com/openguild-labs/polkadot-hackathon-2025

[27] The Ultimate Guide to ZK Coprocessor Technology
https://www.blockchainappfactory.com/zk-coprocessor-solutions

[28] State of Polkadot Q4 2024 - Messari
https://messari.io/report/state-of-polkadot-q4-2024