TLDR; This proposal intends to make a resolver for decoding intents with higher accuracy, also reducing the dependency over LLM for building the transaction object from the intent.

The problem arises with such approach is inaccuracy of calldata/value/amount generated for the intended transaction might not be correct, causing multiple failed transactions which could be prevented by transaction simulation using Alchemy/Tenderly but it seems like a hit and trial method for decoding intents.

Proposed Method

We will take an example intent "Send 10 USDC to 0x118aeFa610ceb7C42C73d83dfC3D8C54124A4946"

This intent will be parsed by the LLM and will return an array of objects based on the number of transactions required to reach the final state as intended.

For the above case it will return :

[{ "contractAddress": "0x742DfA5Aa70a8212857966D491D67B09Ce7D6ec7", "functionId": "0x01", "amount": "0xf", "toAddress": "0x118aeFa610ceb7C42C73d83dfC3D8C54124A4946", "value": "0x" }]

The response object contains the following :

- contractAddress : If the transaction needs to interact with a contract
- functionId : A set of predefined hexes for functions defined below
- amount : Amount of ERC20 tokens if required by the transaction
- toAddress : EOA address or any other address that may be required by a contract call or while sending native tokens
- value : Amount of native token to be send along with the transaction if any

The following hexes will be used for decoding :

- SEND_NATIVE - 0x00
- SEND_ERC20 - 0x01
- SWAP_EXACT_TOKENS_FOR_TOKENS - 0x02
- SWAP_TOKENS_FOR_EXACT_TOKENS - 0x03
- SWAP_EXACT_ETH_FOR_TOKENS - 0x04
- SWAP_ETH_FOR_EXACT_TOKENS - 0x05
- WRAP - 0x06
- UNWRAP - 0x07

Ideal Case :

The response of the LLM is passed on to the resolver which checks the wallet balance, if enough tokens are present for the transaction, it is built and send back to the user for signing the userOp and the gas is sponsored by the Paymaster

Now arises two other edge-cases :

- If the token required for the transaction is not present in the wallet :

In this case the resolver will append one or more of these functions (0x02, 0x03, 0x04, 0x05) before the intended one to swap out the required token and then build batched transactions.

- If the transaction is no way possible through any routes :

The intent is rejected and dropped by the resolver, returning the user a message about the reverted intent

How Nodes are going to execute intents

Intent flow

- User provides the intent in natural language
- Intent is passed through the resolver and intent object is calculated by the LLM and the response is in the format :

[{ "contractAddress": "...", "functionId": "...", "amount": "...", "toAddress": "...", "value": "..." }]

- The intent object is passed to the intent mempool, from where the intents are picked up by the intent builders, they encode the intent transaction routes into a 105 bytes long string as defined below and verified by calling the contract function validateIntentOp
- The intent builders auction for the most efficient route within a defined deadline, once a single intent route is finalised, the userOp

is built and then sent back to the user for signing the userOp and the gas is sponsored by the Paymaster

- Total encoded intent string is 105 bytes long
- 1 Byte : Function Identifier
- Next 20 bytes : To address - used to transfer tokens natively or via contract
- Next 20 bytes : Contract Address to interact with
- Next 32 bytes : Value of Native token
- Next 32 bytes : ERC20 Amount
- 1 Byte : Function Identifier
- Next 20 bytes : To address - used to transfer tokens natively or via contract
- Next 20 bytes : Contract Address to interact with
- Next 32 bytes : Value of Native token
- Next 32 bytes : ERC20 Amount
- For example “Send 123ETH to 56AeF9d1da974d654D5719E81b365205779161aF”

Function (1 byte)

To Address (20 bytes)

Contract Address (20 bytes)

Value(32 bytes)

Amount (32 bytes)

0x00

56AeF9d1da974d654D5719E81b365205779161aF

00000000000000000000000000000000000000000000

00000000000000000000000000000000000000000000000000000000000007b

0000000000000000000000000000000000000000000000000000000000000000

- So for an intent “Send 123ETH to 56AeF9d1da974d654D5719E81b365205779161aF” will be encoded to 0x0056AeF9d1da974d654D5719E81b365205779161aF00000000000000000000000000000000000000000000000000000000000007b0000000000000000000000000000000000000000000000000000000000000

The above proposal of decoding intents works with native token transfer, erc20 token transfers, swap and wrap functionalities, it will be extended to support larger range of transactions and set-up resolvers dedicated for a specific transaction types rather than having a generalised resolver for all.