It is my understanding that calldata length has become a bottleneck for Layer 2 scaling. I have recently begun to collaborate with some people on a possible version 3 of the ABI specification which could help with this.

In v3, a call to foo(bool[12])

would encode as 047f

for [false, false, false, false, false, true, true, true, true, true, true, true]

.

In the header byte, the first three bits are the encoding version identifier, 000

. The next five bits are the function identifier 00100

, in this case, 4. If the function ID is 31 or greater, all five bits are set and the RLP encoding of the function ID is appended to the header byte.

Next is 7f

which is the RLP encoding of the integer which represents the values in the bool array, where each element is represented by a bit. Notice that in this case the first five bools are false

which makes the integer 000001111111

or 127 i.e. 0x7f

.

Boolean arrays are a special case. Everything else is much more normal. Values are straightforwardly encoded as RLP strings, except for arrays and tuples which are encoded as RLP lists.

An example call to foo((string,bool,bool,int72)[2],uint8)

(given function ID 27) is 1bcac44180010ac44201800181ff

.

I would like to know if this would be acceptable for use in Layer 2 and elsewhere. The full implementation in Java is here: GitHub - esaulpaugh/abiv3: ABIv3 proof of concept for Ethereumand I am considering beginning a python implementation.