The following are the details about the Hybrid Query processing point query.

Hybrid Query: The algorithm for querying a given key *Key* in the hybrid query of L2SM is shown in the following Algorithm. First, the first d-2 levels of the L2SM are traversed level by level. A binary search is performed in each level to locate a Bloom filter that may contain the *Key*. Then, the Bloom filter determines whether the *Key* is likely to exist. If the *Key* is likely to exist, the corresponding record is read; if the *Key* does not exist, the search continues to the next level. Repeat the process, and if the *Key* still does not exist when reading the records of level d-2, locate the index interval in level d-1 that contains the *Key* according to the records in level d-2, with the help of the left and right pointers established by L2SM. Read all the records in the interval, if it contains the *Key*, then return the record; if the interval does not contain the *Key*, it means that the record does not exist.

Algorithm Hybrid Query of L2SM for Processing Point Query

Input: Target Key Key, Root Node $root$, Depth d

if Key exists in RAM or $root$ **then**
    Get KV pair directly and return
end if
for Layer $i = 0$ to $d - 2$ **do**
    for Each SSTable SST in Layer i **do**
        if Bloom filter of SST gives positive feedback **then**
            Read SSTable SST and return KV pair if Key in SST'
        end if
    end for
end for
$SST \leftarrow$ Read SSTable in Level $d - 2$ whose key range includes Key
$index \leftarrow$ Get the index of first Key less than Key in SST
$LP, RP \leftarrow$ Get left and right pointer by the $index$ and $index + 1$
$\mathbf{K} \leftarrow$ Get KV pairs bounded by LP and RP
if Key in $\mathbf{K}$ **then**
    Return KV pair
else
    Return NULL

The following is a discussion of the complexity of L2SM implementation and maintenance.

Implementation and maintenance. We discuss the complexity of implementation and maintenance in terms of the code workload, code maintenance, and L2SM application. First, the code workload of L2SM is comparable to that of LSM. In our implementation, we made full use of existing libraries and frameworks which significantly simplify the process of implementing the storage structure and optimization algorithms. Second, we focused on the modularity and maintainability of the code when designing the L2SM. We divided the system into multiple independent modules, each focusing on a specific function. This not only improves the code readability but also makes it easier to maintain. Finally, we have completed the design and implementation of L2SM with comprehensive testing and validation. It can be directly applied to real-world environments. Therefore, the increase in complexity of implementation and maintenance is negligible.