```
Traceback (most recent call last)
in <module>:23

  20 )
  21
  22 # Inspect a sample:
❱ 23 sampled_data = next(iter(train_loader))
  24
  25 print("Sampled mini-batch:")
  26 print("===================")

/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:630 in __next__

   627 │   │   │   if self._sampler_iter is None:
   628 │   │   │   │   # TODO(https://github.com/pytorch/pytorch/issues/76750)
   629 │   │   │   │   self._reset()  # type: ignore[call-arg]
❱  630 │   │   │   data = self._next_data()
   631 │   │   │   self._num_yielded += 1
   632 │   │   │   if self._dataset_kind == _DatasetKind.Iterable and \
   633 │   │   │   │   │   self._IterableDataset_len_called is not None and \

/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:673 in _next_data

   670 │
   671 │   def _next_data(self):
   672 │   │   index = self._next_index()  # may raise StopIteration
❱  673 │   │   data = self._dataset_fetcher.fetch(index)  # may raise StopIteration
   674 │   │   if self._pin_memory:
   675 │   │   │   data = _utils.pin_memory.pin_memory(data, self._pin_memory_device)
   676 │   │   return data

/usr/local/lib/python3.10/dist-packages/torch/utils/data/_utils/fetch.py:55 in fetch

  52 │   │   │   │   data = [self.dataset[idx] for idx in possibly_batched_index]
  53 │   │   else:
  54 │   │   │   data = self.dataset[possibly_batched_index]
❱ 55 │   │   return self.collate_fn(data)
  56

/usr/local/lib/python3.10/dist-packages/torch_geometric/loader/link_loader.py:211 in collate_fn

  208 │   │   r"""Samples a subgraph from a batch of input edges."""
  209 │   │   input_data: EdgeSamplerInput = self.input_data[index]
  210 │   │
❱ 211 │   │   out = self.link_sampler.sample_from_edges(
  212 │   │   │   input_data, neg_sampling=self.neg_sampling)
  213 │   │
  214 │   │   if self.filter_per_worker:  # Execute `filter_fn` in the worker process

/usr/local/lib/python3.10/dist-packages/torch_geometric/sampler/neighbor_sampler.py:334 in
sample_from_edges

  331 │   │   inputs: EdgeSamplerInput,
  332 │   │   neg_sampling: Optional[NegativeSampling] = None,
  333 │   ) -> Union[SamplerOutput, HeteroSamplerOutput]:
❱ 334 │   │   out = edge_sample(inputs, self._sample, self.num_nodes, self.disjoint,
  335 │   │   │   │   │   │     self.node_time, neg_sampling)
  336 │   │   if self.subgraph_type == SubgraphType.bidirectional:
  337 │   │   │   out = out.to_bidirectional()

/usr/local/lib/python3.10/dist-packages/torch_geometric/sampler/neighbor_sampler.py:666 in
edge_sample

  663 │   │   │   │   │   input_type[0]: torch.cat([src_time, dst_time], dim=0),
  664 │   │   │   │   }
  665 │   │
❱ 666 │   │   out = sample_fn(seed_dict, seed_time_dict)
  667 │   │
  668 │   │   # Enhance `out` by label information ##################################
  669 │   │   if disjoint:

/usr/local/lib/python3.10/dist-packages/torch_geometric/sampler/neighbor_sampler.py:431 in
_sample

  428 │   │   │   │   num_sampled_nodes = num_sampled_edges = None
  429 │   │   │
  430 │   │   │   else:
❱ 431 │   │   │   │   raise ImportError(f"'{self.__class__.__name__}' requires "
  432 │   │   │   │   │   │   │   │     f"either 'pyg-lib' or 'torch-sparse'")
  433 │   │   │
  434 │   │   │   if num_sampled_edges is not None:

ImportError: 'NeighborSampler' requires either 'pyg-lib' or 'torch-sparse'
```