Repo	Defect Name/ID/Link	Defect Description	Scenario Analysis	HEM	Confidence	ODC
Pytorch	Running torch.sort can corrupt memory #11189 https://github.com/pytorch/pytorch/issues/111189	Calling the torch.sort function will result in a crash/memory corruption in cases where sorting 1-dimensional tensors with a sufficient amount of elements would cause out-of-bounds access. This function works in multi-dimensional traversals.	The edge case of 1-dimensional tensor was not properly handled.	HEM2	High	Checking
Pytorch	torch.dynamo.exc.Unspported:unexpected sourceless type bases #110315 https://github.com/pytorch/pytorch/issues/110315	Program throws “Unsupported” exception during JIT compilation, reading Keyed-Jagged-Tensors constructed within the module as “sourceless”. This is due to the Dynamo (JIT) compiler extracting data from KJTs without defining the source.	Developers did not define the source before using the JIT compiler.	HEM5	Med	Assignment
Pytorch	pytree.tree_map does not respect type of torch.Size #111962 https://github.com/pytorch/pytorch/issues/111962	The tree_map function accepts torch.Size() data types, which is an array of tuples, but attempts to call methods that tuple does not have. This causes unexpected or bad behavior.	Function attempts to invoke tuple methods on the array of tuples, rather than the tuple elements.	HEM1		Function
Pytorch	Different behaviors on torch.var_mean with torch.compile(default mode)	Resolving division by 0 happens differently depending on compilation type. "Eager mode", which executes operators as soon as they are encountered) gives NaN result and Inductor (Just-In-Time compiler that resolves operators as needed when the code is executed) returns 0. This caused errors in variance corrections when the number was larger than that of data points in the model.	Developers assumed general rule for division by zero, when each compiler type resolves it differently.	HEM3	Med	Interface
Pytorch	Compilation failure of torch.nn.functional.adaptive_max_pool3d_with_indices in torch.compile Optimized mode. #112496 https://github.com/pytorch/pytorch/issues/112496	Compilation of this function fails in optimized mode, because tensors are converted to Symbolic tensors when optimizing, which are not properly handled by the function (expects non-symbolic tensors).	Part of the optimization process involves converting tensors to symbolic tensors, which causes compilation failure when functions expecting non-symbolic tensors are invoked.	HEM6	Med	Build/Package/Merge
Pytorch	Passing non-contiguous inputs to SDPA on CUDA device with mem-efficient attention backend returns garbage #112577 https://github.com/pytorch/pytorch/issues/112577	Scaled_dot_product in Pytorch 2.0.1 used the math() function and 2.1 uses mem_efficient_attention, which does not perform the operation the same way and returns garbage.	When updating to a new version, developers changed the underlying arithemtic function for function, which caused a defect in surrounding code.	HEM6	High	Build/Package/Merge
Pytorch	_foreach_add segfaults when passed tensors with different lengths #112305 https://github.com/pytorch/pytorch/issues/112305	_for_each_add accepts multiple tensors but does not verify their sizes before performing the add operation. If tensor 1 is size n and tensor 2 is size n+1, an out-of-bounds-memory access will occur on the final iteration, causing a segfault.	Developers did not implement proper bounds checking before iteration.	HEM2	High	Checking
Pytorch	torch.jit.optimize_for_inference assumes forward method #108662 https://github.com/pytorch/pytorch/issues/108662	The torch.jit.opitmize_for_inference allows for passing of functions to specify methods/attributes to optimize. This includes optimizing a module's .forward() method. All modules do not include this method() and will cause errors when it attempts to optimize a method that does not exist.	Verification that disparate modules have .forward() method was not done before relying on it within invoking method.	HEM5	Med	Interface
Pytorch	Segfault in ‘flatbuffer_loader.cpp:298’ #109793 https://github.com/pytorch/pytorch/issues/109793	The code does not verify the flatbuffers module has all fields initialized, allowing the passing of bad/null values as input, which causes a null pointer error in the underlying C++ code when accessing those values leading to a segfault.	Failure to encode pre-conditions when calling C++ functions in the outer Python code.	HEM2	High	Interface
Pytorch	torch.finfo(torch.flat8_e4m3fn).max crashes python runtime #109737 https://github.com/pytorch/pytorch/issues/109737	torch.finfo() looks for a "max" member, which does not exist in floats, creating an unhandled exception trying to access a thing that doesn't exist leading ot a segfault.	Developers wrote function relying on the given data type's "max" member to prevent iteration out of bounds, but accepts floats which do not have this member.	HEM1	High	Checking
Pytorch	CrossEntropy fails silently #117532 https://github.com/pytorch/pytorch/issues/117532	CrossEntropy allows the passing of invalid classes (does not validate input), which causes an out of bounds memory access and crash.	Cases with bad input were not tested.	HEM6	Med	Function
Pytorch	torch.foreachmul_segmentation fault #113156 https://github.com/pytorch/pytorch/issues/113156	A segfault occurs when enumerating over for-each multiplication but not for-each division, due to multiplication calling mul.out, which performs what ends up being out-of-bounds memory access, while division uses div_.Tensor, which does not do this.	Division calls a safe underlying function to perform its arithmetic, where as multiplication uses a different underlying function that behaves differently.	HEM1		Function

Pytorch	Unexpected reshard in backward because of unused gradable input in frozon modules #117510 https://github.com/pytorch/pytorch/issues/117510	The _reshard function is not idempotent as it needs to be, so calling it several times produces different results and states, in turn leading to errors due to attempts at resharding happening before gradable input is unsharded, throwing an error due to memory allocation.	Execution of function produces different output from same input, thus when called repeatedly, causing additional function calls, and rapidly consuming all the alloted memory producing an error.	HEM4	Low	Algorithm

Pytorch	Dynamo fails to track dataclass #116264 https://github.com/pytorch/pytorch/issues/116264	Code expects from torch.vmap to be available inside torch.compile do not actually exist: Using the line
		var.call_method(tx, "__init__", args, kwargs)
		return var
		  else:
		the _init_ method call of the object creation is never called, and the initial data is not correctly stored in “var”.
			Called function does not exist in module causing unexpected value to be stored in variable.	HEM3	Med	Interface

Pytorch	DDPOptimizer lazy compile causes shape mismatch error #116300 https://github.com/pytorch/pytorch/issues/116300	Compiling with lazy mode in the distributed training module produces stride/shape mismatch errors in tensors that do not appear in other compliation modes. This appears to occur because compile_check_fn is used to check dynamic dimensions of tensors, but with optimization, the call of this function is delayed. Because dynamo uses a guard that expects by default dim[0]=200, this will likely not be satisfied at the point the guard makes its check	Optimization changes the execution order of code, causing a necessary function call to occur after code that depends on it.	HEM9	Med	Build/Package/Merge
Pytorch	Wrong result when operator.abs is called after abs #117757 https://github.com/pytorch/pytorch/issues/117757	Absolute value function not working correctly every loop. Ex: In the second iteration of the loop, the call to opt_fn generates wrong results tensor([-2., -2., -2., -2.]) VS tensor([2., 2., 2., 2.]). This could be because of optimization issues, but one commentor suggested the abs(negative) is being dropped before being emitted to the underlying C++ for some reason that has to do with SymInts during the optimization process	Behavior of abs() function with ints is not same as behavior of abs() function with symbolic ints.	HEM1	Med	Interface
Pytorch	assert_size_stride bug in inductor generated code #115344 https://github.com/pytorch/pytorch/issues/115344	Compiling with one kernel (Triton) does not satisfy expected arguments when "autotuning", and thus produces out-of-bounds memory access errors. This appears to be caused by the cvmm_triton function being called with different input shapes, creating errors that are not handled. Other compilation modes use a different function, which does handle these errors.	Different compliation modes use different functions to compile code, with differing error-checking. Code was not properly tested across differing compilation modes.	HEM6	Med	Build/Package/Merge
Pytorch	The static checks of the TransformerEncoder should consider num_layers to avoid IndexError. #10335 https://github.com/pytorch/pytorch/pull/103335	TransformerEndcoder causes an IndexError when num_encoder_layers is set to 0. The code works when num_encoder_layers is set to something greater than 0, but attempts an out of bounds access if set to 0.	Pre-condition of greater than 0 was not encoded, with error handling for 0-case.	HEM2	High	Checking
Pytorch	Fix Python-bound function signature (torch._C.Graph.addInput) #88528 https://github.com/pytorch/pytorch/pull/88528	Function signatures do not match between addinput(self, name: str) -> value and function call addInput(const std::string& name =””). Seems to be an error when the Python is compiled to C++, due to incorrect inputs.	Developers assumed function signatures were the same between the Python on C++ code.	HEM1	Med	Build/Package/Merge
Pytorch	Add unit test for nested_tensor input to nn.TransformerEncoder. # 100650 https://github.com/pytorch/pytorch/pull/100650	Test coverage did not adequately cover inputs to TransformerEncoder, causing errors.	Insufficient test coverage.	HEM6	High	Checking
Pytorch	[cuDNN][cuDNN V8 API] Use suggest memory format for cuDNN V8 API #87617 https://github.com/pytorch/pytorch/pull/87617	Observed failures in funtorch tests resulting from benchmark cache collisions due to incorrect memory format. Memory format being dependent on both input and weight would resolve this error.	Developers failed to properly define requirements for memory format.	HEM2	High	Interface
Pytorch	Align mask formatting of both masks more closely #96286 https://github.com/pytorch/pytorch/pull/96286	Inconsistent formatting of canonical masks at various points in the TransformerDecoder causes errors when the boolean masks are passed as input.	Function does not define correct types of acceptable input.	HEM2	Med	Function

Pytorch	Handle trailing masked column behavior for nested tensor #100113 https://github.com/pytorch/pytorch/pull/100113	Unexpected behavior occurs in trailing masked column behavior with nested tensors when "enable_nested_tensor" is set to true, removing the column of output when there is only 1. This causes inconsistency in output, especially when using aggregation functions. Behavior should be consistent for edge cases.	Edge case of single column output not tested.	HEM6	Med	Checking
Pytorch	[pthreadpool] Set max threadlimit to tsan limit #89453 https://github.com/pytorch/pytorch/pull/89453	Max thread limit causing an internal assert due to exceeding the tsan (thread sanitizer/safety check) on clang. It is falling to 64, and the cap is 63.	Failure to encode appropriate upper bound to input, causing out of bounds access.	HEM2	High	Checking
Pytorch	MHA torch.jit.script fix for in_proj_weight = None #95653 https://github.com/pytorch/pytorch/pull/95653	JIT compiler does not properly handle cases where in_proj_weight field is set to "None".	Compiler requires parameter to have a value, does not encode this rule.	HEM2	Med	Checking
Pytorch	Segfault in new_empty_strided #82416 https://github.com/pytorch/pytorch/issues/82416	Input validation does not occur, leading new_size.size() to not equal new_stride.size() in the TORCH_CHECK() function. This leads to a seg fault.	Bounds-checking fails due to lack of input validation.	HEM2	High	Checking
Pytorch	is_causal parameter in torch.nn.TransformerEncoderLayer.forward does not work #96941	When "is_causal" is set to True, the program overrides the use of the "attention" mask. This causes undefined behaviors when the input is not a causal mask, including certain methods such as torch.nn.TransformerEncoderLayer.forward not working.	Function behaves unexpectedly in cases with certain parameters. Inadequate testing to discover this.	HEM6	Med	Assignment
Pytorch	Accuracy minifier can find spurious accuracy failures involving uninitialized memory #93437 https://github.com/pytorch/pytorch/issues/93437	When declaring a new graph, if there is no explicitly initialized data in it, the data that already existed in the unitialized memory location will cause a difference in the given and expected results when minifying the graph, including inaccurate results. Initialization should either set an initial value or handle uninitialized graphs.	Graph declaration fails to clear previous data from graph, causing inaccurate results when nodes are not explicitly overwritten. Developers should have encoded this rule.	HEM2	High	Checking
Pytorch	outputs can require grad even when compiled in no_grad region #115294 https://github.com/pytorch/pytorch/pull/115294	The auto-differentiating gradient function was not being applied correctly, which created mismatched outputs (combining inputs in unexpected ways), and incorrectly required gradients even when compiled in "no gradient" regions.	Function rules for handling inputs were not properly encoded, creating unexpected and inaccurate outputs.	HEM2	Med	Function
Pytorch	Inductor gives obscure error when FX graph to be compiled returns tuple #93593 https://github.com/pytorch/pytorch/issues/93593	Inductor compiler throws an assert error when attempting to compile an FX graph that returns a tuple. It turned out that this was caused by the minifier- the contract with one of the underlying functions to minify a graph accepted only a specific type of input.	Underlying functions worked with other inputs, but did not work with tuples.	HEM1	Med	Checking
Pytorch	nn.Transformer contains unsupported tensor scalar type #85116 https://github.com/pytorch/pytorch/issues/85116	Pytorch throws a RuntimeError for "unexpected tensor scalar type" when using torch.onnx.export to export a module containing an nn.Transformer module. ONNX is a open-source format for exporting deep learning modules, and the code that converts the model to that format (whether in exporter or nn.Transformer itself) is not properly handling the data per the standard.	Rules for formatting ONNX modules were not properly encoded, causing errors in invoking code that expects correctly formatted data.	HEM2	Med	Algorithm
Pytorch	Scripted reshape incorrect if shape is dynamically calculated #78721 https://github.com/pytorch/pytorch/issues/78721	"When exporting using torch.onnx.export, torch.onnx.export produces incorrect export of reshape function call after scripting if shape is calculated dynamically. It looks like one of the shape arguments is not converted to integer and is float instead. function" Essentially, the results of integer and float arithemtic are different, but the export function treats them interchangeably when shape is calculated dynamically.	Function treats integers and functions interchangably, when their behavior needs to be defined separately.	HEM2	Med	Algorithm
Pytorch	Diffuser pipeline device attribute broken when using optimized model #93602 https://github.com/pytorch/pytorch/issues/93602	When using a compiled model in a DDMPipeline, the self.device property will always return "cpu" instead of the correct value. This is due to the optimized model class not inheriting from nn.Module	Developers invoked module expecting it to inherit members from a parent class, but the inheritance was not properly encoded.	HEM2	Low	Interface

Pytorch	Minifier crash #93613 https://github.com/pytorch/pytorch/issues/93613	Minifier crashing due to expected inputs being incorrect. The issue is that input mismatches occur when parts of the graph are replaced, as it assumed that order was preserved in placeholder node lists but it was not. The issue was solved with a dictionary to keep 1:1 mapping between the placeholders and original inputs.	Developers failed to ensure 1:1 ordered mapping between placeholder data and new inputs.	HEM2	Med	Checking
Pytorch	Dynamo can not trace 'int(a_scalar_tensor.item())' #93515 https://github.com/pytorch/pytorch/issues/93515	The underlying expect_true in the tracing process doesn't work with integer variables but not floats. The issue was resolved by casting all floats to int before comparing them.	Invoked function expects integers, but the code was allowing floats to be passed.	HEM1	High	Checking
Pytorch	PyTorch Embedding Op with max_norm is not working as expected #81717 https://github.com/pytorch/pytorch/issues/81717	When using a GPU with multidimensional input, the renormalization when max_norm is specified will renormalize to the wrong value (less than max_norm), due to being reliant on different functions. The behavior of this operation should be consistent whether performed with CPU or GPU.	Behavior when normalizing with GPU should behave the same as behavior when normalizing using CPU.	HEM1	Med	Function
Pytorch	autograd: handle detach() and no_grad() mutations on input #95980 https://github.com/pytorch/pytorch/pull/95980	The issue arises from functions that mutate leaves on graphs, but are relying on calling other functions that rely on auto-gradient, where leafs are immutable. The solution was to perform mutation before passing to autograd functions.	Developers expected rule of mutability in graph nodes to apply to invoked functions utilizing immutable inputs.	HEM1	Med	Interface
Pytorch	A Simple Function Causing Graph Break #93486 https://github.com/pytorch/pytorch/issues/93486	Unsupported function call error raised when using torch._dynamo.optimize in certain contexts. This error was caused by the sub-function transformers.is_torch_tpu_available() using func.tools.lru_cache, which was not supported in these contexts, but was in others.	Called function works in certain contexts, but not in others.	HEM1	Med	Interface
Pytorch	tensordot not working for dtype int32 and lower when there is only 1 element in the given axis #84530 https://github.com/pytorch/pytorch/issues/84530	Tensor dot product function (tensordot()) not working correctly with dInts of size 32 and lower. Only works with 64 bit long integers. Code was using a function that returned int64, but allowed the user to specify another type causing a mismatch.	Developers did not encode requirement for explicitly defined data type.	HEM2	Med	Checking
Pytorch	[ONNX] Use topk to export max(dim,keepdim) to onnx #76344 https://github.com/pytorch/pytorch/issues/76344	When exporting the max() function, the code was calling both max() and argmax() which creates issues when exporting in the ONNX format runtime. The topk() function achieves a similar purpose without causing issues.	Rules for ONNX runtime were not encoded in modules that export to it.	HEM2	High	Interface