2026 春季人工智能大赛赛题报告

1. 总体架构

实现由四层组成：

- **抽象层：** `Generator` 句柄与 `GeneratorImpl` 多态接口，统一管理 seed、state、device 和 mutex；
- **后端层：** `CPUGeneratorImpl` 管理 `std::mt19937` 状态， `CUDAGeneratorImpl` 管理 Philox seed/subsequence；
- **默认状态层：** CPU 维护全局默认 Generator，CUDA 按 device 维护独立默认 Generator， `manual_seed(uint64_t)` 统一重置默认状态；
- **算子层：** `Uniform` 、 `Normal` 、 `KaimingUniform` 、 `Rand` 、 `Randn` 和普通 `Dropout` 接收显式 Generator，缺省时使用目标设备默认 Generator。

CPU/CUDA 随机分布与 Dropout 支持 `FLOAT16` 、 `BFLOAT16` 、 `FLOAT32` 、 `FLOAT64` 。当前不覆盖 AlphaDropout、FeatureDropout、inplace Dropout 和 CUDA Graph RNG 状态管理。

2. Generator / GeneratorImpl 模块

本章介绍随机数基础设施的核心抽象，包括 Generator Handle、GeneratorImpl 多态接口、默认 Generator 管理及辅助接口，并分析其与 PyTorch Generator 的接口语义关系。该模块主要完成以下内容：

- Generator Handle 与 GeneratorImpl 多态架构；
- Generator 生命周期、seed 与 state 管理；
- CPU/CUDA 默认 Generator 管理；
- 工厂函数、类型检查和默认选择辅助接口；
- Generator 与 PyTorch 接口语义对齐分析。

2.1. 实现概述与接口对齐说明

Generator 模块负责随机数系统的状态管理，不直接绑定某一种随机算法。整体职责如下：

- `Generator` ：用户侧可持有的轻量句柄，对外提供统一的 seed、state、device 与 clone 接口；
- `GeneratorImpl` ：后端实现的抽象基类，定义统一随机数状态接口；
- `CPUGeneratorImpl` / `CUDAGeneratorImpl` ：CPU 与 CUDA 后端的具体实现；
- Default Generator：维护 CPU 全局状态和各 CUDA 设备的独立默认状态；

- 工厂与辅助接口：负责 Generator 创建、默认 Generator 获取、后端检查和显式/默认 Generator 选择。

随机算子不直接依赖 `std::mt19937` 、 `curand` 或 Philox 的实现细节，而是先接收可选的 `Generator` 句柄，再由 `get_generator_or_default` 解析为对应后端实现。未传入 Generator 时，算子根据目标 Tensor 的设备选择默认 Generator；显式传入时优先使用用户指定状态。这使随机数消费者与底层随机数生成器解耦，也为后续接入其他设备后端预留了扩展位置。

相关核心实现在以下文件中：

- `infini_train/include/generator.h`
- `infini_train/src/generator.cc`
- `infini_train/src/core/runtime/cpu/cpu_generator_impl.{h,cc}`
- `infini_train/src/core/runtime/cuda/cuda_generator_impl.{h,cc}`

`Generator` 持有 `std::shared_ptr<GeneratorImpl>` 。复制 Generator 时复制的是句柄，两个句柄共享同一个随机状态；需要独立状态时使用 `clone()` 深拷贝 Impl。默认构造的 `Generator` 为 undefined 状态，与 PyTorch `at::Generator` 的默认构造语义一致，不能作为已初始化的随机源使用。

2.2. 状态与线程安全设计

seed 用于初始化随机序列，state 则表示随机序列已经推进到的位置。一次随机算子调用会推进 Generator 的内部状态，因此连续调用不会重复得到同一段随机序列；通过保存并恢复 state，可以准确重放保存点之后的随机结果。

GeneratorImpl 提供 `mutex_` 。CPU 随机 kernel 在整个生成循环期间持锁；CUDA kernel 在预留 Philox subsequence 时持锁，随后将 seed 和 subsequence 作为 kernel 参数传入设备端。全局 `manual_seed` 也会在重置默认状态时持锁。该策略遵循 PyTorch 的调用方持锁惯例：Generator 不承诺所有公开状态操作自动线程安全，调用方在需要组合多个状态读写操作时可通过 `Generator::mutex()` 建立临界区。

2.3. 默认 Generator 与全局种子

CPU 后端通过 `getDefaultCPUGenerator()` 返回进程内唯一的默认 CPU Generator。CUDA 后端通过 `getDefaultCUDAGenerator(device_index)` 返回指定设备的默认 Generator，每个设备对应独立状态来源。默认 Generator 首次获取时使用非确定性随机种子初始化，后续获取同一设备时返回同一状态来源。

框架级 `manual_seed(uint64_t seed)` 依次重置 CPU 默认 Generator 和所有可用 CUDA 设备的默认 Generator。这样，参数初始化、 `Rand/Randn` 和 Dropout 在相同 seed、相同设备、相同调用顺序下可以稳定复现。

2.4. 与 PyTorch 的对齐与差异

基本一致的行为如下：

- Handle 与 Impl 分层，Generator 句柄共享底层状态；
- `set_current_seed` 、 `current_seed` 、 `seed` 、 `get_state` 、 `set_state` 、 `device` 、 `clone` 的职责一致；
- 默认 CPU Generator 为全局状态，CUDA 默认 Generator 按设备维护；
- 显式 Generator 优先，未传入时选择默认 Generator；
- Generator 类型不匹配时拒绝将 CPU Generator 用于 CUDA 后端，或反向使用；
- 随机状态访问采用公开 mutex、调用方在状态操作处持锁的方式。

当前未完全对齐的部分如下：

1. 智能指针类型不同。InfiniTrain 使用 `std::shared_ptr` ，PyTorch 使用 `c10::intrusive_ptr` ，因此没有 PyTorch 中 `unsafeReleaseGeneratorImpl()` 一类的所有权转移接口。
2. CPU state 的内部格式不同。InfiniTrain 基于 `std::mt19937` 的流序列化结果和正态缓存构造 `UINT8` Tensor state；该格式可在同一构建环境恢复，但不承诺跨标准库实现或跨编译器版本稳定。
3. 当前未实现 PyTorch 的 CUDA Graph graph-safe state 接口。CUDA Graph 尚未进入框架范围，因而没有 `graphsafe_get_state()` 、 `graphsafe_set_state()` 及 capture 专属随机状态。
4. 未实现 PyTorch 的全部随机分布、全部随机算子和全部 Dropout 变体。

3. CPU 后端、CUDA 后端与随机算子接入

本章介绍 CPU Generator 与 CUDA Generator 的状态组织、随机数消费方式，以及初始化和训练期随机算子如何接入统一 Generator 接口。CPU 与 CUDA 共享 Generator 抽象，但不要求同一 seed 下产生逐元素一致的随机值。

3.1. 后端设计概览

能力	CPU Generator	CUDA Generator
底层状态	`std::mt19937` engine 与正态缓存	seed 与 Philox subsequence
随机数消费位置	CPU host kernel	CUDA device kernel
默认 Generator	进程内全局实例	每个 CUDA device 一个实例
state 保存恢复	支持	支持
连续状态推进	engine 每次取样推进	每次 kernel 预留 subsequence

能力	CPU Generator	CUDA Generator
设备随机 kernel	CPU distribution kernel	`curandStatePhilox4_32_10_t` kernel

3.2. CPU Generator 实现

CPU 后端使用 `std::mt19937` 作为核心随机引擎。`CPUGeneratorImpl` 保存初始 seed、Mersenne Twister engine 状态，以及 float/double 正态分布的 Box-Muller 缓存样本。重设 seed 时会清空两种正态缓存并重新初始化 engine，避免旧缓存影响新序列。

CPU 分布 kernel 使用 `uniform_real_distribution` 与 `normal_distribution` 辅助实现。Uniform 会通过 `random()` 或 `random64()` 取原始随机比特并映射至目标区间；Normal 使用 Box-Muller 变换，在可用时保存第二个样本以减少随机数消耗。该设计确保 Normal 的缓存也会被 state 保存和恢复。

CPU state 以 CPU `UINT8` Tensor 返回。`set_state` 会检查 state 的 device、dtype、最小长度、缓存标志以及 `std::mt19937` 反序列化结果；全部验证通过后才更新 Generator，避免部分非法输入损坏原状态。

3.2.1. 与 PyTorch 基本一致的行为

- 都使用 Mersenne Twister 系列随机引擎；
- 支持 seed 重设、非确定性 seed、默认 Generator、clone 与 state 恢复；
- Normal 的缓存属于 Generator 状态的一部分，重设 seed 时会清空缓存；
- CPU 随机 kernel 在使用 engine 时持有 Generator mutex。

3.2.2. 与 PyTorch 的差异

- PyTorch 使用自行实现的 `at::mt19937`，而当前实现使用 C++ 标准库的 `std::mt19937`；
- PyTorch 的 CPU state 是稳定的内部 POD 表示，当前 CPU state 使用标准库流序列化，因此跨构建兼容性较弱；
- CPU 端没有公开的 offset 接口。CPU engine 按抽样推进内部状态，外部无需直接操作 offset。

3.3. CUDA Generator 实现

CUDA Generator 保持与 CPU 相同的 seed/state/clone/device 接口，但内部状态采用适合并行 kernel 的 Philox 模型。`CUDAGeneratorImpl` 保存 `seed_` 和 `next_philox_subsequence_`。每次 Uniform、Normal 或 Dropout CUDA kernel 启动前，主机端在 mutex 保护下为该调用预留一段 subsequence；kernel 中每个元素由 `seed + subsequence + index` 初始化独立的 `curandStatePhilox4_32_10_t` 状态。

这种方式避免了不同 kernel 调用重复使用同一随机序列区间，也不会要求 GPU kernel 修改共享 Generator 对象。CUDA state 由 seed 和 subsequence 构成，并以 CPU `UINT8` Tensor 保存；恢复

state 后，后续 CUDA 随机调用会从相同 subsequence 继续执行。

3.3.1. 与 PyTorch 基本一致的行为

- CUDA 默认 Generator 按 device 维护，彼此状态独立；
- seed 与 Philox counter/subsequence 共同表示 CUDA RNG 的推进状态；
- 支持显式 seed、非确定性 seed、state 保存恢复与 clone；
- CUDA kernel 在设备端使用 Philox/curand 状态生成随机数；
- 同一 seed、同一设备和同一调用顺序下可复现。

3.3.2. 暂未对齐的语义及原因

- PyTorch 使用共享的 `CUDAGeneratorState` 管理更复杂的 CUDA Graph capture 状态；当前实现仅维护 Generator 私有 seed/subsequence 状态；
- 未实现 PyTorch 对 CUDA Graph capture 期间随机状态操作的限制与 graph-safe state；
- 当前 subsequence 分配面向已接入的分布和 Dropout kernel，尚未覆盖 PyTorch 的全部随机消费模型；
- 多 GPU 默认 Generator 已实现，尚未覆盖分布式训练中全部跨 rank 随机数策略。

3.4. 初始化、随机张量与 Dropout 接入

初始化层的 `Uniform` 、 `Normal` 、 `KaimingUniform` 接收可选 Generator，并根据 Tensor 的设备经 dispatch stub 选择 CPU 或 CUDA kernel。 `Rand` 和 `Randn` 在函数式接口中创建指定 shape、dtype、device 的 Tensor 后，分别复用 Uniform 和 Normal 初始化路径。

普通 Dropout 同时接入 CPU 和 CUDA。前向阶段随机生成 `UINT8` mask，并对保留元素乘以 `1 / (1 - p)` ；反向阶段使用前向保存的同一 mask 计算梯度，不生成新的随机数，也不额外推进 Generator。Dropout 支持显式 Generator；未传入时自动使用输入 Tensor 所在设备的默认 Generator。

CPU 与 CUDA 随机分布、Dropout 均支持 `FLOAT16` 、 `BFLOAT16` 、 `FLOAT32` 、 `FLOAT64` 。对非浮点 dtype 的随机初始化和 Dropout 会进行参数/类型检查，避免以错误的存储布局写入数据。

4. 测试与验证

当前测试覆盖 Generator 接口、state 恢复、默认/显式 Generator、初始化、随机张量和 Dropout 等核心路径；完整训练工作流的端到端随机测试仍可作为后续补充。

4.1. 测试内容

1. **接口与状态测试**：验证 CPU/CUDA Generator 的 device、seed、state、默认 Generator 获取，以及 state 的 `UINT8` CPU Tensor 约定。

2. **种子可复现性测试**：验证 Uniform、Normal、KaimingUniform、Rand、Randn 和 Dropout 在相同 seed 下可复现，在不同 seed 下发生变化。
3. **状态恢复测试**：保存 Generator state，生成下一段随机数，恢复 state 后重新生成，验证两段结果相同。
4. **默认与显式 Generator 测试**：验证显式 Generator 不推进默认 Generator；未传 Generator 时目标设备默认状态被推进。
5. **多设备默认状态测试**：在至少两张 GPU 的环境中，验证 `cuda:1` 随机调用只推进 `cuda:1` 默认 Generator。单 GPU 环境会按条件跳过该测试。
6. **Autograd 兼容性验证**：完整构建验证远端 FunctionCtx 重构后，Exp 与 Dropout 的保存状态逻辑能够正确接入新的 Autograd 上下文；Exp 反向回归测试已执行。Dropout 的正式 Generator 测试当前重点覆盖前向随机性与可复现性，独立的反向数值断言可作为后续补充。

相关测试文件包括：

- `tests/generator/test_generator.cc`
- `tests/generator/test_generator_interface.cc`
- `tests/generator/cuda_only/test_cuda_generator.cc`
- `tests/autograd/test_autograd_elementwise_backward.cc`

4.2. 本地复现命令

以下命令已在 WSL Ubuntu 22.04、CUDA 环境下从新的构建目录实际执行。当前工程的 CUDA 源文件需要 C++20 标准库，因此使用 GCC 13，并显式指定 NVCC 的 host compiler 同为 GCC 13； `CMAKE_POLICY_VERSION_MINIMUM` 用于兼容项目引入的旧版 gflags CMake 配置。无 CUDA 环境可将 `USE_CUDA` 设为 `OFF` 并省略 `CMAKE_CUDA_HOST_COMPILER` ，CPU Generator 测试仍可执行。

```
git submodule update --init --recursive
cmake -S . -B build-report-repro \
  -DBUILD_TEST=ON \
  -DUSE_CUDA=ON \
  -DUSE_NCCL=OFF \
  -DCMAKE_POLICY_VERSION_MINIMUM=3.5 \
  -DCMAKE_C_COMPILER=gcc-13 \
  -DCMAKE_CXX_COMPILER=g++-13 \
  -DCMAKE_CUDA_HOST_COMPILER=/usr/bin/g++-13
cmake --build build-report-repro -j8
ctest --test-dir build-report-repro \
  -R 'Generator|test_generator' \
  --output-on-failure -j1
```

完整回归可使用：

```
ctest --test-dir build-report-repro --output-on-failure -j4
```

```
(base) chaos@DESKTOP-9FD9T17:~/projects/Infini/InfiniTrain/build-test-full-gcc13$ ctest -R 'Generator|test_generator' --output-on-failure -j1
Test project /home/chaos/projects/Infini/InfiniTrain/build-test-full-gcc13
      Start 293: CPU/GeneratorTest.SupportsDeviceAndSeedInterface/1-byte object <00>
 1/19 Test #293: CPU/GeneratorTest.SupportsDeviceAndSeedInterface/1-byte object <00> .....................   Passed    0.05 sec
      Start 294: CPU/GeneratorTest.UniformIsReproducibleForSameSeed/1-byte object <00>
 2/19 Test #294: CPU/GeneratorTest.UniformIsReproducibleForSameSeed/1-byte object <00> .....................   Passed    0.04 sec
      Start 295: CPU/GeneratorTest.NormalIsReproducibleForSameSeed/1-byte object <00>
 3/19 Test #295: CPU/GeneratorTest.NormalIsReproducibleForSameSeed/1-byte object <00> .....................   Passed    0.04 sec
      Start 296: CPU/GeneratorTest.RandAndRandnSupportExplicitAndDefaultGenerators/1-byte object <00>
 4/19 Test #296: CPU/GeneratorTest.RandAndRandnSupportExplicitAndDefaultGenerators/1-byte object <00> .....   Passed    0.15 sec
      Start 297: CPU/GeneratorTest.DropoutMaskIsReproducibleForSameSeed/1-byte object <00>
 5/19 Test #297: CPU/GeneratorTest.DropoutMaskIsReproducibleForSameSeed/1-byte object <00> ................   Passed    0.14 sec
      Start 298: CPU/GeneratorTest.ConsecutiveCallsAdvanceSequence/1-byte object <00>
 6/19 Test #298: CPU/GeneratorTest.ConsecutiveCallsAdvanceSequence/1-byte object <00> .....................   Passed    0.04 sec
      Start 299: CPU/GeneratorTest.DifferentSeedsProduceDifferentSequences/1-byte object <00>
 7/19 Test #299: CPU/GeneratorTest.DifferentSeedsProduceDifferentSequences/1-byte object <00> .............   Passed    0.04 sec
      Start 300: CPU/GeneratorTest.SameSeedAndCallOrderReproduceResults/1-byte object <00>
 8/19 Test #300: CPU/GeneratorTest.SameSeedAndCallOrderReproduceResults/1-byte object <00> ................   Passed    0.14 sec
      Start 301: CPU/GeneratorTest.StateRestoreReplaysUniformSequence/1-byte object <00>
 9/19 Test #301: CPU/GeneratorTest.StateRestoreReplaysUniformSequence/1-byte object <00> .................   Passed    0.04 sec
      Start 302: CPU/GeneratorTest.StateRestoreReplaysNormalSequence/1-byte object <00>
10/19 Test #302: CPU/GeneratorTest.StateRestoreReplaysNormalSequence/1-byte object <00> ..................   Passed    0.04 sec
      Start 303: CPU/GeneratorTest.ExplicitGeneratorDoesNotAdvanceDefaultGenerator/1-byte object <00>
11/19 Test #303: CPU/GeneratorTest.ExplicitGeneratorDoesNotAdvanceDefaultGenerator/1-byte object <00> .....   Passed    0.14 sec
      Start 304: CPU/GeneratorTest.MissingGeneratorUsesAndAdvancesDefaultGenerator/1-byte object <00>
12/19 Test #304: CPU/GeneratorTest.MissingGeneratorUsesAndAdvancesDefaultGenerator/1-byte object <00> .....   Passed    0.14 sec
      Start 305: CPU/GeneratorTest.RepeatedDefaultGeneratorLookupSharesState/1-byte object <00>
13/19 Test #305: CPU/GeneratorTest.RepeatedDefaultGeneratorLookupSharesState/1-byte object <00> ...........   Passed    0.04 sec
      Start 306: CPU/GeneratorTest.GlobalManualSeedReproducesDefaultGeneratorResults/1-byte object <00>
14/19 Test #306: CPU/GeneratorTest.GlobalManualSeedReproducesDefaultGeneratorResults/1-byte object <00> ...   Passed    0.13 sec
      Start 307: CPU/GeneratorTest.KaimingUniformUsesExplicitGenerator/1-byte object <00>
15/19 Test #307: CPU/GeneratorTest.KaimingUniformUsesExplicitGenerator/1-byte object <00> .................   Passed    0.04 sec
      Start 308: GeneratorInterfaceTest.CpuStateIsAnOpaqueUint8CpuTensor
16/19 Test #308: GeneratorInterfaceTest.CpuStateIsAnOpaqueUint8CpuTensor ..................................   Passed    0.04 sec
      Start 309: GeneratorInterfaceTest.CpuStateRoundTripRestoresSeed
17/19 Test #309: GeneratorInterfaceTest.CpuStateRoundTripRestoresSeed .....................................   Passed    0.04 sec
      Start 310: test_generator_cuda
18/19 Test #310: test_generator_cuda .......................................................................   Passed    0.31 sec
      Start 311: test_generator_cuda_only
19/19 Test #311: test_generator_cuda_only ..................................................................   Passed    0.14 sec

100% tests passed, 0 tests failed out of 19

Label Time Summary:
cpu     =   1.27 sec*proc (17 tests)
cuda    =   0.45 sec*proc (2 tests)

Total Test time (real) =   1.73 sec
```

```
267/270 Test #236: CPU/TransformerModuleTest.GPT2TransformerLayer/1-byte object <00> ..........................***Skipped   0.05 sec
268/270 Test #237: CPU/TransformerModuleTest.GPT2Model/1-byte object <00> ....................................***Skipped   0.04 sec
269/270 Test #240: CPU/TransformerModuleTest.StateDict/1-byte object <00> ....................................***Skipped   0.04 sec
270/270 Test #238: CPU/TransformerModuleTest.LLaMA3Model/1-byte object <00> ..................................***Skipped   0.04 sec

100% tests passed, 0 tests failed out of 267

Label Time Summary:
cpu     =  25.25 sec*proc (233 tests)
cuda    =  14.74 sec*proc (11 tests)

Total Test time (real) =  11.11 sec

The following tests did not run:
          1 - demangle (Disabled)
         25 - dcheck_on (Disabled)
         26 - dcheck_off (Disabled)
        116 - CPU/AutogradElementwiseBackwardTest.BFloat16MulBroadcastBackwardLargeBlock/1-byte object <00> (Skipped)
        230 - CPU/TransformerModuleTest.Embedding/1-byte object <00> (Skipped)
        231 - CPU/TransformerModuleTest.LayerNorm/1-byte object <00> (Skipped)
        232 - CPU/TransformerModuleTest.RMSNorm/1-byte object <00> (Skipped)
        233 - CPU/TransformerModuleTest.GPT2MLP/1-byte object <00> (Skipped)
        234 - CPU/TransformerModuleTest.SwiGLUMLP/1-byte object <00> (Skipped)
        235 - CPU/TransformerModuleTest.StandardAttention/1-byte object <00> (Skipped)
        236 - CPU/TransformerModuleTest.GPT2TransformerLayer/1-byte object <00> (Skipped)
        237 - CPU/TransformerModuleTest.GPT2Model/1-byte object <00> (Skipped)
        238 - CPU/TransformerModuleTest.LLaMA3Model/1-byte object <00> (Skipped)
        240 - CPU/TransformerModuleTest.StateDict/1-byte object <00> (Skipped)
```

在本次最终合并后的 CPU+CUDA 构建环境中，上述命令从干净目录完成配置与编译。Generator 相关的 19 个 CPU/CUDA 测试全部通过；完整 CTest 共 270 项，其中 256 项通过，3 项为禁用测试，11 项因特定 Transformer 配置或其他测试条件而跳过，无失败项。

4.3. 本章小结

测试结果表明，Generator 状态在初始化、随机张量和 Dropout 调用之间能够连续推进；全局 seed 和显式 state 恢复均可稳定控制随机行为；CPU 与 CUDA 后端通过同一 Generator 接口提供一致的使用方式。当前覆盖已满足赛题要求的基础设施验证，后续可继续补充训练脚本级端到端可复现性测试和更多随机算子测试。

5. 附录

5.1. Generator / GeneratorImpl 接口与 PyTorch 对齐表

InfiniTrain 接口或机制	PyTorch 对应接口/概念	作用	对齐情况
`Generator`	`at::Generator`	用户持有的随机状态句柄	基本一致
`GeneratorImpl`	`c10::GeneratorImpl`	后端多态实现基类	基本一致
`set_current_seed(seed)`	`set_current_seed(seed)`	设置当前 seed 并重置随机序列	一致
`current_seed()`	`current_seed()`	获取当前 seed	一致
`seed()`	`seed()`	使用非确定性 seed 重置并返回 seed	一致
`get_state()`	`get_state()`	返回 CPU Byte state	基本一致
`set_state(state)`	`set_state(state)`	恢复随机状态	基本一致
`device()`	`device()`	查询 Generator 所属设备	一致
`mutex()`	`mutex()`	提供随机状态同步锁	一致， 调用方持锁
`clone()`	`clone()`	复制当前随机状态到独立 Generator	一致
`make_generator`	`make_generator`	创建指定 Impl 的 Generator	一致
`check_generator`	`check_generator`	校验后端类型并转为具体 Impl	一致

InfiniTrain 接口或机制	PyTorch 对应接口/概念	作用	对齐情况
`get_generator_or_default`	`get_generator_or_default`	选择显式或默认 Generator	一致
`getDefaultCPUGenerator`	`getDefaultCPUGenerator`	获取 CPU 默认 Generator	一致
`getDefaultCUDAGenerator`	`getDefaultCUDAGenerator`	获取指定 CUDA 设备默认 Generator	一致
`manual_seed(seed)`	`at::manual_seed` / `torch.manual_seed`	重置 CPU 与 CUDA 默认 Generator	基本一致
CUDA graph-safe state	`graphsafe_get_state` / `graphsafe_set_state`	CUDA Graph 随机状态切换	暂未实现
CPU/CUDA offset 公共接口	`set_offset` / `get_offset`	直接管理部分后端 offset	暂未提供

5.2. 参考资源说明

本项目参考 PyTorch 的 Generator 抽象、CPU/CUDA Generator、随机分布与 Dropout 设计。具体参考文件、链接、代码位置、许可证和独立修改说明将按比赛诚信守则写入与本报告一同提交的 `REFERENCE.md` 。

5.3. 后续工作

- 扩展更多随机分布、随机采样算子和 Dropout 变体；
- 增加模型训练脚本级的端到端可复现性测试；
- 支持 CUDA Graph graph-safe RNG state；
- 研究稳定的跨编译器 CPU RNG state 表示；
- 根据后端扩展需求补充更多设备的 GeneratorImpl。