Contents

Generator 抽象选题设计报告 赛题未报名

本 PR 实现 2026 春大赛训推赛道 T2-2-1 中的 Generator 抽象设计。由于该任务报名人数限制，未进入正式参赛名单，但仍尝试遵循任务要求做了一些设计，完成核心架构抽象与部分后端落地。

下表列出了任务书主要功能要求与当前实现之间的对应关系，便于快速确认完成情况。

任务书要求	完成情况	对应章节
统一 Generator 抽象设计	✅	第 2 章
CPU Generator 实现	✅	第 3.1 章
CUDA Generator 实现	◐	第 3.2 章
默认 Generator 管理机制	✅	第 2 章
统一全局随机种子入口	✅	第 2 章
Generator State 获取与恢复	✅	第 2、3 章
初始化随机算子接入	✅	第 4 章
训练随机算子（Dropout）接入	✅	第 4 章
Generator 与 PyTorch 接口语义对齐分析	✅	第 2～4 章
功能测试与可复现性验证	✅	第 5 章

本项目围绕随机数生成基础设施（Generator）进行了整体设计与实现，参考 PyTorch Generator 架构，当前完成情况如下。

模块	完成情况	说明
Generator 抽象设计	✅	完成 Generator Handle、GeneratorImpl 多态架构，实现统一随机数接口抽象。
CPU Generator	✅	完成 Seed、State、Clone、默认 Generator、序列化等核心能力。
CUDA Generator	◐	完成 Generator 状态管理与接口实现，Device RNG Kernel、CUDA Graph 等高级能力暂未实现。
默认 Generator 管理	✅	支持 CPU/CUDA 默认 Generator、统一获取入口及 ManualSeed 管理。
随机算子接入	✅	完成 Uniform、Normal 等初始化算子及 Dropout 等训练随机算子的 Generator 接入。
PyTorch 对齐分析	✅	分析当前实现与 PyTorch 的接口语义一致性及差异。
测试与验证	✅	完成接口、Seed、State、默认 Generator、随机算子等核心测试。

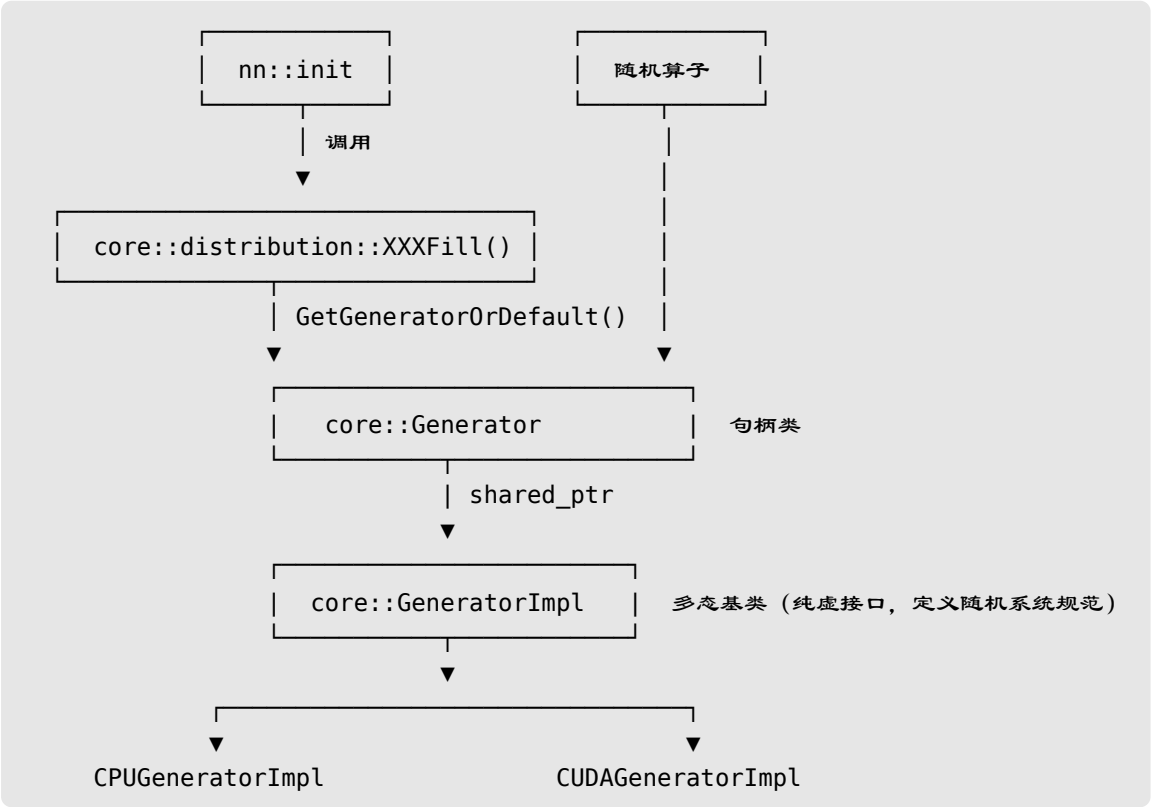
- ◐：完成 Generator 基础设施及接口设计，部分功能（如 CUDA Device RNG Kernel、CUDA Graph 等）暂未实现，即 CUDA 后端功能尚未完全实现。

1. 架构说明

本章介绍 Generator 基础设施的整体架构设计。首先给出随机数系统整体组成及各模块之间的关系，随后介绍 Generator 抽象层、CPU/CUDA 后端、默认 Generator 管理机制以及随机算子接入方式。整体设计参考 PyTorch Generator 架构，在保证接口语义一致的基础上，为后续随机分布扩展、CUDA Kernel 接入及更多设备后端预留统一接口。

本项目参考 PyTorch Generator 的整体设计，在当前框架中构建了统一的随机数生成基础设施。整体采用 Generator Handle + GeneratorImpl 多态实现的分层结构，并分别实现 CPU 与 CUDA 后端 Generator，使随机算子能够通过统一接口访问随机数状态，而无需关心底层设备实现。

架构图如下：



从整体流程来看，Generator 负责统一管理随机数状态，Distribution 负责随机分布组织，CPU/CUDA Backend 分别完成底层随机数生成，实现了随机数消费者与随机数实现之间的解耦。具体各模块的实际实现在下述展开。

2. Generator/GeneratorImpl 模块

本章介绍 Generator 基础设施的核心抽象设计，包括 Generator Handle、GeneratorImpl 多态接口、默认 Generator 管理及相关辅助接口，并分析当前实现与 PyTorch Generator 在接口语义上的对应关系，主要完成以下内容：

- Generator Handle 与 GeneratorImpl 多态架构；
- Generator 生命周期及状态管理；
- 默认 Generator 管理机制；
- 工厂函数及辅助接口；
- 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 模块负责整个随机数系统中的状态管理，不直接负责随机数生成。整体职责如下：

- Generator：用户持有的随机数句柄，对外提供统一接口；
- GeneratorImpl：随机数生成器抽象基类，定义统一随机数接口规范；
- CPUGeneratorImpl / CUDAGeneratorImpl：不同设备后端的具体实现；
- Default Generator：维护各设备默认随机数状态；
- 工厂函数：负责 Generator 创建、默认 Generator 获取及类型检查。

随机算子仅通过 Generator 获取随机数状态，而无需关心底层随机数实现，从而实现随机数消费者与随机数生成器之间的解耦。 `Generator` 负责管理随机数生成器的状态，提供用户接口，并持有一个指向多态基类 `GeneratorImpl` 的智能指针。用户通过该句柄类可以方便地调用不同平台（CPU/GPU）的随机数生成器实现，而无需关心底层细节。相关实现在 `infini_train/src/core/generator.h` 与 `infini_train/src/core/generator.cc` 中。 具体接口设计与 PyTorch 对齐情况可参考附录

2.2. 对齐说明与未对齐分析

1. 基本一致的行为：
   Handle 语义对齐、生命周期管理类似（shared_ptr / intrusive_ptr）、多态 Impl 层、Seed 管理语义对齐、默认 Generator 管理、Clone 语义、Philox 偏移、设备类型校验与 Copy/Move 删除

2. 不一致的行为
   1. 智能指针类型不一致（std::shared_ptr , c10::intrusive_ptr）（导致不支持 unsafeGeneratorReleaseImpl()）
   2. State 表示方式不同（std::vector , c10::TensorImpl）
   3. handle 层的锁机制不同 （ PyTorch 中 GeneratorImpl.h#L33-L47）
      - 当前实现对变更方法自动加锁，PyTorch 中由调用方负责、需手动加锁
   4. State 自描述格式使用 magic+version 提供跨版本兼容检查
   5. 此外，还有部分涉及 CUDA Graph、Dispatcher 的功能暂未实现。

3. CPU 后端与 GPU 后端

本章介绍 CPU Generator 与 CUDA Generator 的具体实现方式，包括各自维护的随机数状态、状态管理方式及与 PyTorch 的语义对齐情况。由于 CPU 与 CUDA 后端在随机数生成机制上存在较大差异，因此分别进行介绍，并说明当前实现范围及后续扩展方向。

3.1. 后端设计概览

CPU 与 CUDA Generator 采用统一接口、不同状态组织方式的设计。

能力	CPU Generator	CUDA Generator
底层随机引擎	std::mt19937_64	Philox Counter
状态组成	Engine State	(Seed, Offset)
随机数生成位置	CPU	Host（当前实现）
默认 Generator	支持	支持
状态恢复	支持	支持
Device RNG	无需	暂未实现

3.2. CPU Generator 实现思路

CPU Generator 基于 C++ 标准库提供的 `std::mt19937_64` 实现随机数生成，并由 GeneratorImpl 统一管理随机状态。随机算子在生成随机数时直接访问 Generator 内部随机引擎，因此 CPU Generator 同时负责随机数状态维护与随机数生成。

由于 CPU 后端不存在类似 CUDA Kernel 的随机数消费模型，因此当前实现无需维护额外 Offset，仅保存随机引擎状态即可实现随机序列恢复。 CPU 后端使用 C++ 标准库中的 `std::mt19937_64` 引擎作为随机数生成器的核心。继承自 GeneratorImpl，接口实现同见附录。

3.2.1. 基本一致的行为：

引擎类型相似(均基于 Mersenne Twister 算法)、种子管理、偏移拒绝、默认种子与 Clone 语义。

3.2.2. 暂未对齐的语义：

- 由于当前 normal 没有缓存机制，目前 set_current_seed()也没有清空缓存的操作，而 PyTorch 中会清空缓存。
- CurrentSeed()返回的是独立的 seed_，而 PyTorch 中，由于实用的是自定义的 at::mt19937 引擎，内部对种子有维护。
- GetOffset()在当前实现中固定返回 0，而 PyTorch 中，会直接 throw 异常，提示 CPU 不支持偏移。
- 使用随机数引擎是 std::mt19937_64，而 PyTorch 中使用的是自定义的 at::mt19937，内部实现略有差异。
- 当前运行内核代码直接操作引擎（需持有 mutex_），而 PyTorch 是值拷贝的形式。

3.2.3. 待实现功能：

- 接入 normal 分布采样缓存后，需在 set_current_seed()中清空缓存。

- 如有必要，可对齐 PyTorch 在 CPU 后端实现随机整数的快捷接口 `random()/random64()`

3.3. CUDA 后端

CUDA Generator 与 CPU Generator 保持统一接口，但由于 CUDA 随机数通常由 Kernel 在 Device 侧消费，因此当前实现主要完成 Generator 状态管理及 Philox 状态维护。真正的 Device RNG Kernel 暂未实现，因此当前 CUDA Generator 更侧重于 Generator 抽象及随机状态管理，而非随机数生成本身。即 CUDA 后端的开发为半成品，接口见附录。

3.3.1. 基本一致的行为

- Generator 生命周期与 CPU Generator 保持一致，均支持 Clone、GetState、SetState 等基础接口。
- CUDA 侧同样支持显式 Seed 设置与随机 Seed 初始化。
- Generator 状态均可通过序列化与反序列化进行保存与恢复。
- 当前实现已经支持 Philox Counter Based RNG 的核心语义，即 `(seed, offset)` 状态组合。

3.3.2. 暂未对齐的语义及原因

- PyTorch CUDA Generator 基于 `CUDAGeneratorState` 共享状态对象实现，而当前实现采用 Generator 私有状态，因此 `graphsafe_get_state()`、`graphsafe_set_state()` 暂无法实现。
- PyTorch 要求 Philox Offset 始终满足 `offset % 4 == 0`，这是由于 `Philox4x32` 每次消费 128bit 随机块；当前实现尚未加入该约束。
- PyTorch 在 CUDA Graph Capture 期间禁止部分状态修改操作，并维护 Capture 专属 RNG 状态；当前项目尚未引入 CUDA Graph，因此相关逻辑缺失。
- PyTorch 的默认 Generator 为「每 GPU 一个全局实例」，当前实现仍以显式构造 Generator 为主。
- PyTorch 中 `philox_cuda_state()` 已成为 Kernel 侧标准接口，而 `philox_engine_inputs()` 属于兼容过渡接口；当前项目仍采用后者设计。

3.3.3. 当前未实现能力

主要包括三类：
- Device RNG Kernel
- CUDA Graph 相关支持
- 多 GPU 高级状态管理

上述功能均建立在当前 Generator 抽象之上，后续可在不修改上层接口的情况下逐步补充。

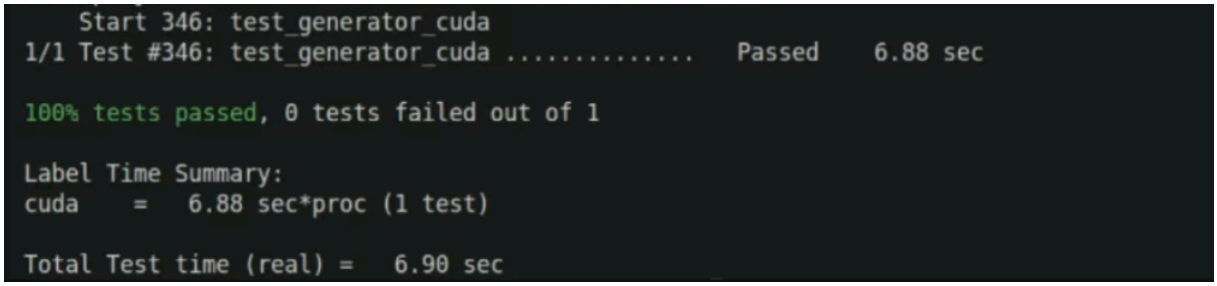
3.3.4. 本章小结

CPU Generator 与 CUDA Generator 共用统一 Generator 抽象及接口规范，但根据不同设备特点采用不同状态组织方式。CPU 后端直接维护随机引擎状态，负责随机数生成；CUDA 后端维护 Philox 随机状态，为 Device 随机数生成提供基础。当前已完成 Generator 基础设施及状态管理，后续可进一步扩展 Device RNG、CUDA Graph 及多 GPU 支持，而无需修改 Generator 上层接口。

4. 测试与验证

单元测试代码在 pr 中涵盖，不是很严谨且覆盖面不完整，缺少完整的训练中测试。本地测试命令如下（无多卡环境）

```
mkdir build && cd build
cmake -DBUILD_TEST=ON -DUSE_CUDA=ON -DUSE_NCCL=OFF ..
make -j
ctest -R "test_generator" --output-on-failure
```



5. 附录

5.1. Generator/GeneratorImpl 模块接口设计与 PyTorch 对齐分析

函数名	对应接口	作用	对齐
SetCurrentSeed(seed)	set_current_seed(seed)	设置当前随机数生成器的种子值	✅
ManualSeed(seed)	set_current_seed(seed)	设置当前随机数生成器的种子值	✅ 别名
CurrentSeed()	current_seed()	获取当前随机数生成器的种子值	✅
Seed()	seed()	自动播种并返回	✅
SetOffset(offset)	set_offset(offset)	设置当前随机数生成器偏移量，用于控制随机数序列的起始位置	✅
GetOffset()	get_offset()	获取当前随机数生成器的偏移量	✅
GetState()->vector()	get_state()->TensorImpl	获取当前随机数生成器的状态	🔶有差异
SetState(vector)	set_state(TensorImpl)	设置当前随机数生成器的状态	🔶有差异
GetDevice()	device()	获取当前随机数生成器所在的设备信息	✅
Mutex()	mutex()	获取当前随机数生成器的互斥锁对象，用于线程安全操作	✅
UnsafeGetImpl()	unsafeGetGeneratorImpl()	获取当前随机数生成器的底层实现对象指针（不安全操作）	✅
Get()	get()	获取当前随机数生成器的底层实现对象指针（安全操作）	✅
Clone()	clone()	克隆当前随机数生成器，返回一个新的 Generator 对象	✅
GetGeneratorOrDefault	get_generator_or_default	获取当前随机数生成器的底层实现对象指针，如果不存在则返回默认 Generator 对象	✅
MakeGenerator	make_generator	创建一个新的随机数生成器对象，并返回其 Generator 句柄	✅
CheckGenerator	check_generator	验证一个 Generator 对象并转换为具体的后端形态，必须提供一个已定义的生成器	✅ 内部接口
detail::DefaultCPUGenerator()	getDefaultCPUGenerator()	获取进程级默认 CPU Generator 单例	✅
detail::DefaultCUDAGenerator()	getDefaultCUDAGenerator()	获取指定 CUDA 设备对应的默认 Generator	✅
ManualSeed(seed) （全局）	torch.manual_seed(seed)	设置全局默认 Generator 的种子值	✅
GetDefaultGenerator(device)	getDefaultGenerator(device)	根据 Device 自动选择 CPU 或 CUDA 默认 Generator	✅
DefaultManualSeed()	default_rng_seed_val （内部实现）	记录最近一次全局 ManualSeed 的值，用于后续懒加载 Generator 初始化	🔶 差异
MutableDefaultCPUGenerator()	default_cpu_generator()	获取可修改的默认 CPU Generator 实例	✅ 内部接口
MutableDefaultCUDAGenerator()	default_cuda_generator()	获取可修改的默认 CUDA Generator 实例	✅ 内部接口

5.2. CPU 后端 Generator 接口设计与 PyTorch 对齐分析

函数名	对应 PyTorch 接口	作用	对齐
CPUGeneratorImpl(seed)	CPUGeneratorImpl(seed_in)	构造 CPU 随机数生成器，并初始化随机引擎状态	✅
SetCurrentSeed(seed)	set_current_seed(seed)	设置当前随机数生成器种子，并重置随机序列	🔶
CurrentSeed()	current_seed()	获取当前随机数生成器种子值	🔶
Seed()	seed()	使用非确定性随机源重新播种，并返回新的 seed	✅
SetOffset(offset)	set_offset(offset)	设置随机序列偏移量；CPU 后端仅允许 offset=0	✅
GetOffset()	get_offset()	获取随机序列偏移量；CPU 后端固定返回 0	🔶
GetState() -> vector	get_state() -> Tensor	获取当前随机数生成器完整状态，用于恢复随机序列	🔶
SetState(vector)	set_state(Tensor)	恢复随机数生成器状态，实现随机序列回滚	🔶
GetDevice()	device()	获取当前 Generator 所属设备信息	✅
Clone()	clone()	复制当前 Generator 及其随机状态，生成独立随机流	✅
Engine() -> mt19937_64&	engine()	获取底层 mt19937_64 随机引擎引用，供随机算子直接使用	🔶
kDeviceType	device_type()	声明当前 Generator 对应设备类型为 CPU	✅
kDefaultSeed	default_rng_seed_val	默认随机种子常量	✅

5.3. CUDA 后端 Generator 接口设计与 PyTorch 对齐分析

函数名	PyTorch 实现	当前实现	说明
`CUDAGeneratorImpl`	支持 device_index、支持共享 state 构造	支持 device_index 与 seed 初始化	当前实现仅维护 seed 与 philox_offset，未引入共享状态对象
`SetCurrentSeed`	更新 seed 并重置 Philox Offset	已实现	行为基本一致
`CurrentSeed`	返回当前 seed	已实现	行为一致
`Seed`	生成非确定性随机种子并调用 set_current_seed()	生成随机 seed 并重置 offset	行为一致
`SetOffset`	设置 Philox Offset，同时要求不处于 Graph Capture	直接设置 philox_offset_	当前未增加 Capture 状态检查与 offset 对齐校验
`GetOffset`	返回当前 Philox Offset	已实现	行为一致，但未限制 Capture 期间访问
`GetState`	序列化 seed 与 offset 状态	返回 seed 与 philox_offset 字节流	行为基本一致
`SetState`	恢复 seed 与 offset 状态	支持恢复 seed 与 philox_offset	行为基本一致，暂未兼容旧版状态格式
`GetDevice`	返回 Generator 所属 CUDA Device	已实现	行为一致
`Clone`	深拷贝 Generator State	复制 seed、offset、device_index	行为基本一致，但未共享状态对象
`PhiloxEngineInputs`	返回 (seed, offset)，随后推进 offset	已实现	当前 CUDA Kernel 扩展接口的核心功能
`getDefaultCUDAGenerator`	每个 GPU 维护一个全局默认 Generator	未实现	当前仅支持显式创建 Generator 实例
`createCUDAGenerator`	创建指定设备 Generator	等价于构造函数	后续建议提供工厂接口保持一致
`CUDAGeneratorState::increase`	按 Philox 规则推进 Offset，保证 4 对齐	未实现	当前直接按请求值递增
`philox_cuda_state`	生成 Graph Safe Philox 状态	未实现	依赖 CUDA Graph 支持
`graphsafe_get_state`	返回共享状态对象	未实现	依赖状态对象设计
`graphsafe_set_state`	切换 Generator State 引用	未实现	依赖状态对象设计
`set_philox_offset_per_thread`	设置 Philox Offset 并保证 4 对齐	未实现	当前使用 SetOffset 直接写入
`philox_offset_per_thread`	获取当前 Philox Offset	未实现	当前由 GetOffset 替代