SevenNet + FastEq（sevennet_fasteq）系统分析与设计报告（面向对象视角 + UML）

> 适用范围：本报告面向 `/home/zhouxibo` 下的工程组合（FastEq + SevenNet + LAMMPS/MLIAP），以及 Conda 环境 `sevennet_fasteq` / `sevennet_lmp` / `sevennet_fasteq_new`。
> 课程背景：面向对象程序设计（OOP）。
> 报告目标：结合本次排障与性能优化的聊天过程、日志现象与环境内代码改动，输出一份可向上汇报的"系统分析与设计"文档，重点从 **类职责、对象协作、扩展点设计、UML 类图** 等角度阐述。

---

1. 背景与总体目标

1.1 背景

- SevenNet 在 LAMMPS 的 `pair_style mliap unified` 场景下运行，涉及多层 equivariant convolution（常见 5 层左右）。
- SevenNet 的卷积/张量积核心依赖 cuEquivariance（简称 cueq）及其 PyTorch 前端 `cuequivariance_torch`。
- FastEq 的目标是在不大改上层接口的前提下，提供更快的 fused CUDA kernel（例如 `mptp`、`cwtp`、`stc` 等），并通过"嫁接"方式接入 SevenNet，从而替代/加速 cueq 的算子执行。

1.2 总体目标

1. **正确性优先**：确保在 SevenNet 模型上，FastEq 路径的数值与参考路径（cueq/或 FastEq 的参考实现）一致或在可接受误差范围内。
2. **可控切换**：允许在运行时通过环境变量选择：
    - 走 cueq（原生）；
    - 走 FastEq（替换/加速）；
    - FastEq 下选择分组 / 不分组 / 强制回退；
    - mptp / cwtp / naive 等不同实现路径。
3. **性能优化聚焦**：优先优化 forward（fwd），在达到正确性的前提下减少 kernel 数、改善访存合并、提升 warp 可用性（eligible warps）并降低 stall。

---

2. 系统组成与关键数据流

2.1 组件清单

- **LAMMPS/MLIAP**：驱动 Python 侧模型推理与反向（在最小化/MD 过程中调用）。
    - 入口：`lmp_single/run.sh` + `lmp_single/in.kokkos`。
- **SevenNet（Python）**：模型构建、模块串联、卷积/线性层调用。
    - 关键文件（环境内）：
        - `.../site-packages/sevenn/model_build.py`：根据环境变量/配置决定后端与 patch 行为。

        - `.../site-packages/sevenn/nn/cue_helper.py`：与 cueq/cuequivariance_torch 的 glue 逻辑（本次 FastEq 嫁接重点）。
        - `.../site-packages/sevenn/nn/sequential.py`、`.../site-packages/sevenn/nn/linear.py`：用于调试（打印 shape/irreps）。
- **cueq / cuequivariance_torch（Python）**：
    - `cuequivariance_torch.operations.*`：高层算子包装（Linear、ChannelWiseTensorProduct、FullyConnectedTensorProduct）。
    - `cuequivariance_torch.primitives.SegmentedPolynomial*`：多路径多段多维 tensor product 的通用执行框架。
    - `cuequivariance_torch.primitives.fasteq_segmented_polynomial.py`（仅在 `sevennet_fasteq` 环境存在/使用）：FastEq 的替代实现入口。
- **FastEq（CUDA + PyTorch extension）**：
    - CUDA kernel：如 `FastEq/fasteq/cuda/src/mptp_fwd.cu`、`cwtp_fwd/bwd.cu`、`stc_fwd/bwd.cu` 等。
    - Python op wrapper：如 `FastEq/fasteq/ops/mptp.py`。

2.2 典型数据流（以卷积层为例）

1. SevenNet 构建 descriptor（来自 `cue.descriptors.channelwise_tensor_product(...)` 或 fully connected variant）。
2. `cuequivariance_torch` 将 descriptor 编译/实例化为 `SegmentedPolynomial`（或 `FastEqSegmentedPolynomial`）。
3. forward 中执行：
    - 输入张量（节点特征、边属性、tp 权重等）按 indices/segments 组织；
    - 调用底层 kernel（cueq 的 fused kernel，或 FastEq 的 mptp/cwtp/stc kernel）；
    - 写回输出张量，并继续后续线性层/门控/下一层卷积。
4. backward 中执行：
    - autograd 触发反向；
    - 可能走 FastEq 的 bwd kernel 或回退到通用路径（例如 cwtp_bwd）。

---

3. 环境差异与对照实验设计（sevennet_lmp vs sevennet_fasteq）

3.1 版本对齐情况

实测 `sevennet_lmp` 与 `sevennet_fasteq` 中关键版本一致（示例）：

- Python：3.10.19
- torch：2.7.0+cu126
- sevenn：0.12.0
- cuequivariance_torch：0.8.0

- e3nn：0.5.9

因此，两套环境的差异主要来自 **环境内文件被 patch 的差异**，而非包版本；它们用于“基线 vs 加速”的对照实验。

3.2 sevennet_fasteq 中的关键 patch 点（相对 sevennet_lmp）

`sevennet_fasteq` 相比 `sevennet_lmp`，在 SevenNet 侧被改动的 Python 文件主要集中于：

- `sevenn/model_build.py`：新增读取 `CUEQ_USE_FASTEQ`（或配置 `use_fasteq`）并向下传递 `use_fasteq`。
- `sevenn/nn/cue_helper.py`：引入 FastEq 入口（`cuet.FastEqSegmentedPolynomial`）、打印 descriptor/segments、debug meta 等。
- `sevenn/nn/sequential.py`、`sevenn/nn/linear.py`：加入调试打印（仅在环境变量开启时打印）。

同时，`cuequivariance_torch` 在 `sevennet_fasteq` 环境也被 patch：

- `cuequivariance_torch/__init__.py` 导出了 `FastEqSegmentedPolynomial`。
- `cuequivariance_torch/operations/linear.py`、`tp_channel_wise.py`、`tp_fully_connected.py`：新增 `use_fasteq` 分支，使用 `FastEqSegmentedPolynomial` 替代 `SegmentedPolynomial`。
- `cuequivariance_torch/primitives/fasteq_segmented_polynomial.py`：FastEq 的核心 meta 推导与执行路径（含 grouped / no-grouping 等策略）。

> 重要：`sevennet_lmp` 的 `cuequivariance_torch/primitives/` 中 **没有** `fasteq_segmented_polynomial.py`；因此它天然不会走 FastEq 路径，适合作为 cueq 基线环境。

3.3 对照实验目标与约束（默认视为可跑通）

本项目默认假设 `sevennet_fasteq` 在目标模型上 **可跑通并产出正确结果**。在此基础上，设置 `sevennet_lmp` 作为对照环境的目的不是“验证能否跑通”，而是：

- 对照 **性能**：FastEq 路径（mptp/cwtp/stc）相对 cueq 基线的端到端与算子级耗时收益。
- 对照 **正确性**：FastEq 输出与基线（cueq 或 FastEq 的参考实现）的一致性与误差边界。
- 对照 **可控性**：环境变量开关是否能精确命中预期执行路径（避免脚本覆盖导致的误判）。

约束：无论是否分组/合并 kernel，FastEq 作为替代实现必须满足上层接口契约（shape、indices/segments 语义、数值与 autograd 语义一致）。

---

4. 环境变量与运行开关设计（现状）

4.1 运行脚本中的关键开关

在 `lmp_single/run.sh` 中存在（示例）：

- `CUEQ_USE_FASTEQ`：SevenNet 是否走 FastEq 后端（核心开关）。
- `FASTEQ_ENABLE_MPTP`：FastEq 侧是否允许走 mptp（否则走 cwtp/naive）。
- `FASTEQ_FORCE_GROUPED`：强制 grouped meta。
- `FASTEQ_NO_GROUPING`：禁用分组，走 no-grouping meta。
- `FASTEQ_MERGE_GROUPS`：将 grouped 的多组尽量合并为一次 mptp 调用（减少 kernel 数）。

- `FASTEQ_USE_CWTP_FWD`：强制 mptp fwd 走 cwtp（通常用于正确性/回退）。

4.2 风险点：脚本覆盖用户命令行环境

当前 `run.sh` 里会 `export CUEQ_USE_FASTEQ=1` 之类的默认设置。
这会导致用户在命令行指定 `CUEQ_USE_FASTEQ=0 ./run.sh` 时仍然走 FastEq，从而"误判已回退到 cueq"。

设计建议：
将 `run.sh` 中的 export 改为"尊重外部设置"的形式，例如：

- `: "${CUEQ_USE_FASTEQ:=1}"` 这种"仅当未设置时才赋默认值"；
  或至少在脚本启动时打印最终生效的开关值，避免排障困扰。

---

5. descriptor、分段（segments）与分组（grouping）策略

5.1 非 uniform segments 的来源

在 SevenNet 的某些模型中（对比 MACE 的常见固定 u），descriptor 中 `u` 可能取多个值，例如 `u={32, 64, 128}`。这会导致：

- `uv` operand 可能出现不同段长度（不同 `u`）；
- `iu/jv/kuv` 等 operand 的 segment shape 也随之变化；
- cueq 的 `uniform_1d` 会强制"同一 operand 的 segment shape 必须一致"，因此会触发 assert；
- FastEq 为了覆盖更多模型，需要处理这种 non-uniform segments。

5.2 分组（grouped）与不分组（no-grouping）

- **分组的动机**：将可在同一 kernel/同一 meta 下执行的路径合并，提高内核利用率并减少 launch 次数；对 non-uniform segments，常见策略是按某些维度（例如 u）分桶。
- **不分组的动机**：保持更"通用"的 meta 推导，尽量减少前置限制；但实现难度更高，也更容易导致性能差或维度不一致问题。

在本次日志中出现：

- "uniform 成功率为 0，但仍然 100% descriptor 都分组执行"
  这说明当前实现中的 "grouped" 指的是 **FastEq 的分桶/多次执行策略**，并不等价于 cueq 的 uniform_1d 要求；它是一种"为处理 non-uniform 而不得不分组"的工程路径。

---

6. mptp/cwtp/stc 三类核心算子定位

6.1 cwtp（channel-wise tensor product）

- 适合 channel-wise tensor product（典型 subscripts：`uv,iu,jv,kuv+ijk`）。
- 在 sevennet_fasteq 环境里，
  `cuequivariance_torch.operations.tp_channel_wise.ChannelWiseTensorProduct` 增加了 `use_fasteq` 分支：
  - `self.f` 是原生 `SegmentedPolynomial`（cueq 路径）；

- `self.ff` 是 `FastEqSegmentedPolynomial`（FastEq 路径）。

6.2 mptp（message passing tensor product，含 scatter）

- 目标是把“按边 TP + scatter_sum 到节点”做 fused，提高性能。
- 在 v=1 的七网场景中，mptp 常见瓶颈来自：
  - 非合并访存（例如 “6.6/32 bytes per sector”）；
  - scatter 的原子写冲突（atomic contention）；
  - kernel 数多（按 u 分组导致多次 launch）。

6.3 stc（symmetric tensor contraction）

`FastEq/fasteq/cuda/src/stc_fwd.cu` 显示 stc 有多种 kernel 版本，但当前 launcher 实际调用的是 `stc_fwd_kernel_notiled`：

- 每 block 处理一个 batch b，每线程处理一个 channel j（`blockDim = u`）。
- 使用 shared memory 缓存 `x1` 与 `out_shared`，避免大量 atomicAdd。
- 该 stc kernel **只适用于特定 paths 结构（len=3/4/5）**，不能直接用于任意 cueq descriptor；要用它计算 cueq descriptor，必须先把 descriptor 转换为 stc 可消费的路径表示。

---

7. 性能分析结论（基于已采集的 NCU/日志要点）

7.1 典型瓶颈信号

从 NCU 截图与日志中可归纳：

- **访存合并差**：如 “平均每 sector 仅使用 4.2~6.6 / 32 bytes”，说明 warp 访问地址 stride 大、散乱，导致一次 memory transaction 有效载荷很低。
- **warp 可调度性差**：eligible warps 远小于 active warps，issue slot 利用率低；stall barrier、stall long scoreboard 明显。
- **kernel 数偏多**：按 u 分组常导致单次推理中多次 kernel launch（如 13 个 kernel），launch 开销与 cache 复用被稀释。

7.2 forward 慢于 backward 的合理性讨论

在某些实现中出现 “fwd 明显比 bwd 慢”：

- 若 bwd 回退到更通用但 kernel 数更少的实现（或复用已有中间结果），可能出现 bwd 反而更快的错觉；
- 但通常 TP 场景下 bwd 也很重，因此长期目标仍是同时优化；短期按老师建议优先 fwd 是合理的。

---

8. 接口契约与一致性验证（面向对象“可替换性”护栏）

> 本项目以“系统可跑通”为前提，本节强调：在持续优化（分组、合并 kernel、改变内存布局）过程中，必须引入契约验证作为护栏，确保 FastEq 始终可替换 cueq（LSP）。

8.1 必须维持的契约（Contract）

- **shape/irreps 契约**：每个 `torch.nn.Module` 的输入输出维度与其 `irreps_in/out` 定义一致。
- **descriptor 语义契约**：FastEq 与 cueq 对同一层使用语义等价的 descriptor（即使内部做分组/合并也要保持对外等价）。
- **数值契约**：在设定输入下，FastEq 输出与基线输出误差在阈值内（max_abs/max_rel）。
- **autograd 契约**：反向梯度维度与数值正确，且无越界/非法访问。

8.2 建议的验证机制（可开关）

- **逐层 shape/irreps 检查**：在 `AtomGraphSequential` 遍历模块时，对 `data["x"]` 做维度断言（仅 debug）。
- **算子级 A/B 对比**：在同一 descriptor 下，对比 `SegmentedPolynomial`（cueq）与 `FastEqSegmentedPolynomial` 输出误差。
- **路径命中日志**：打印每层实际命中的后端（cueq / fasteq；mptp / cwtp / naive；grouped / no-grouping），避免脚本覆盖导致误判。

---

9. 推荐的工程化改造方案（面向可维护性）

9.1 接入策略建议：以“语义等价”为第一目标

建议把 FastEq 的接入层做成“可证明等价”的替换：

- 对于同一卷积层，FastEq 与 cueq 必须使用语义等价的 descriptor（包括 flatten/squeeze/split 等变换后的结果）。
- 若 FastEq 需要更原始的结构（例如为了 mptp 的分组），应在内部做映射，但对外保持输出 shape 不变。

9.2 开关策略建议：默认可跑 + 局部替换

推荐默认配置：

- 默认先保证 `CUEQ_USE_FASTEQ=0` 能跑通（基线正确性）。
- FastEq 打开时按层/按算子逐步替换，并提供 “单层回退/单算子回退” 的开关：
  - 例如只替换 cwtp，不替换 linear；或只替换 mptp，不替换 stc。

9.3 测试与验证建议

至少提供两类快速验证：

1. **shape/irreps 一致性检查**（运行时可开关）
   - 每个关键 module 前后打印 `x.shape` 与 `irreps_in/out.dim`，并 assert 对齐（仅 debug）。
2. **数值对比**
   - mptp vs cwtp（同一输入、同一 meta）输出 max_abs/max_rel；
   - 关键点：对比对象要明确（FastEq cwtp 还是 cueq cwtp），并确保两者使用同一 descriptor 语义。

---

10. 当前工作总结（可用于向上汇报）

已完成

- 完成 `sevennet_fasteq` 环境内的 FastEq 接入（SevenNet -> cue_helper -> cuequivariance_torch -> FastEqSegmentedPolynomial）。
- 打通 non-uniform segments 的观察与诊断链路（打印 descriptor、operands、segments、path_indices 等）。
- 增加了可控的 debug 入口（通过环境变量控制输出）。
- 初步完成 mptp 的 fwd/bwd GPU 计时打印（可观察单次 kernel GPU 时间）。
- 收集了 NCU/Profiler 证据，确认主要瓶颈之一是非合并访存（bytes per sector 很低）与 warp stall。

已识别问题

- 在持续优化（分组/合并 kernel/更改布局）过程中，存在 **shape/irreps/descriptor 契约被破坏** 的工程风险；需要用"契约验证护栏"把问题前置发现（详见第 8 节）。
- `run.sh` 中 export 的方式可能覆盖命令行设置，增加排障误导风险。

下一步计划（建议）

1. 持续维护"descriptor/输出语义等价"与契约验证，确保在开启/关闭 FastEq、不同策略组合（grouped/no-grouping，mptp/cwtp/naive）下都保持可替换性。
2. 在正确性护栏完善后，针对 mptp fwd 做访存与 launch 次数优化：
    - 减少分组导致的 kernel 数（同层多 path 合并、减少分桶粒度、或做更稳定的 merge 策略）；
    - 优化 gather 的访存合并（数据布局/索引排序/分桶）；
    - 减少 scatter 的原子写冲突（分块局部累加、warp 聚合、或按 receiver 分块）。

附录 A：常用环境变量速查（建议在脚本打印最终生效值）

- `CUEQ_USE_FASTEQ`：SevenNet 是否走 FastEq 后端。
- `FASTEQ_ENABLE_MPTP`：允许 mptp fused 路径。
- `FASTEQ_FORCE_GROUPED`：强制 grouped meta。
- `FASTEQ_NO_GROUPING`：强制 no-grouping meta。
- `FASTEQ_MERGE_GROUPS`：分组后尝试合并调用。
- `FASTEQ_USE_CWTP_FWD`：mptp fwd 回退到 cwtp（便于对比/保正确）。
- `SEVENN_DEBUG_SEQ`：打印模块前后 `x_shape`（调试）。
- `SEVENN_DEBUG_LINEAR`：打印线性层 `x_shape`/irreps/权重（调试）。

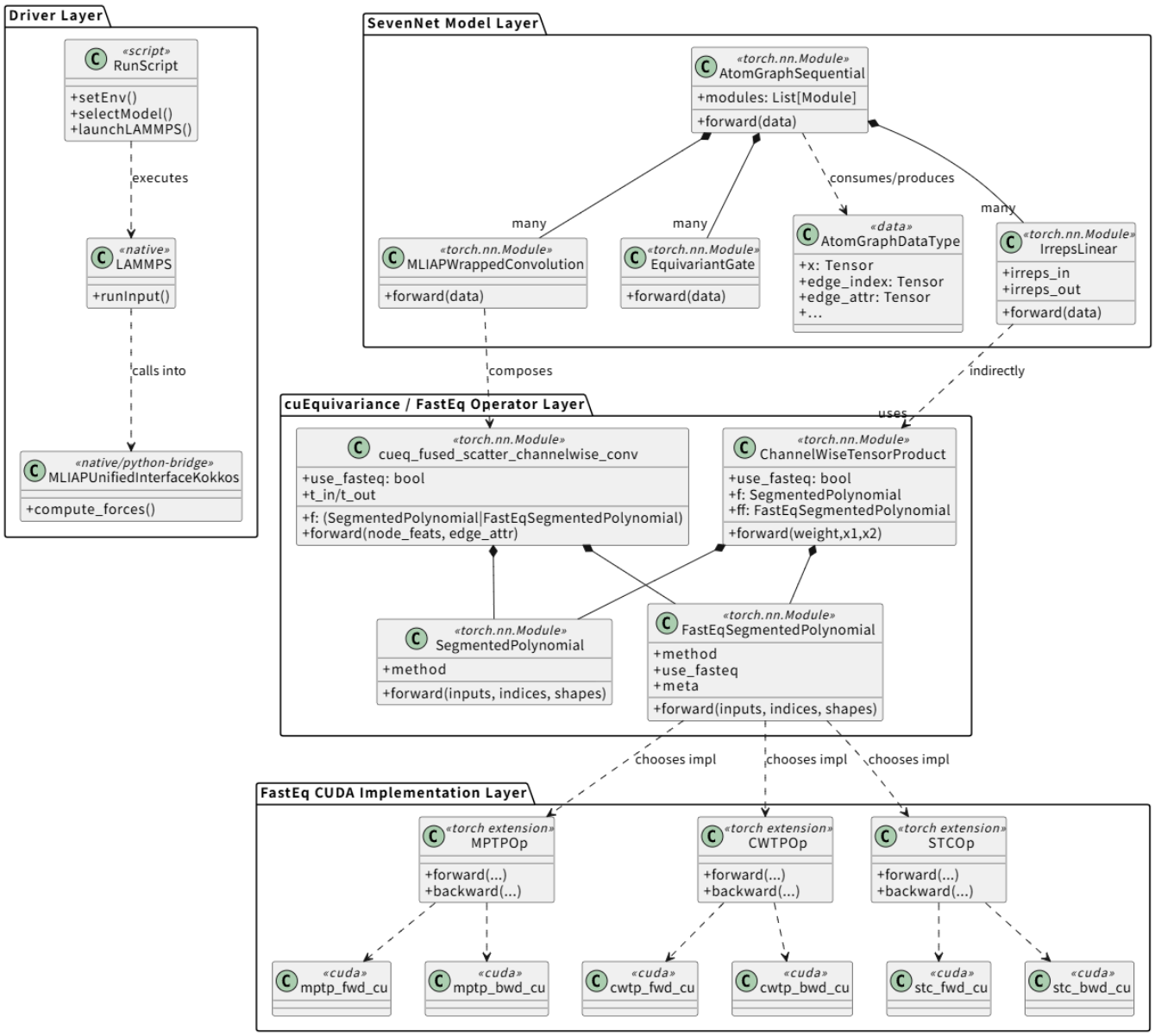
---

3. 面向对象建模：核心类、职责与协作关系（UML）

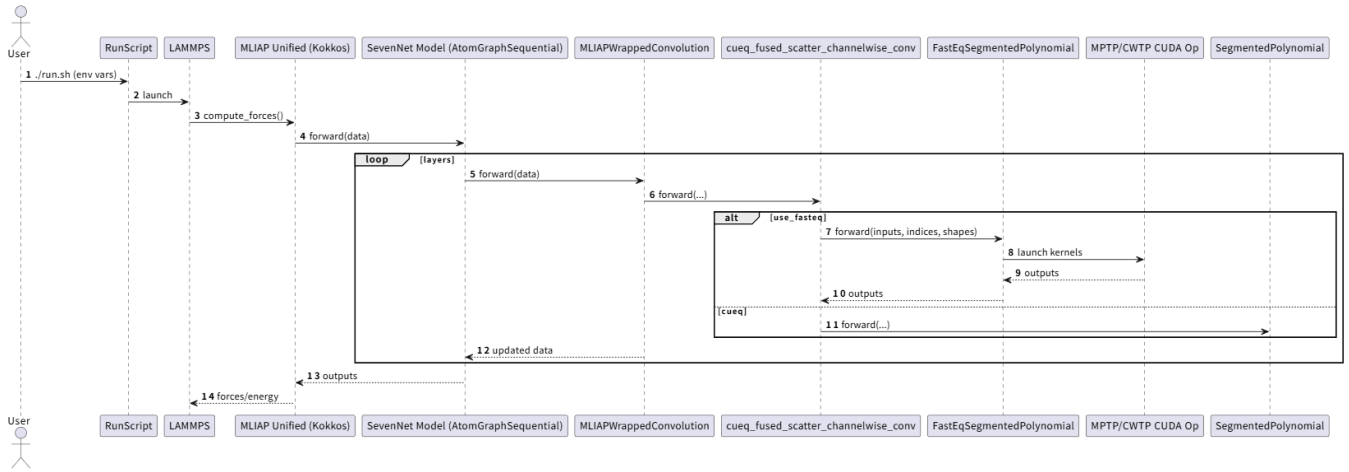
本系统可以按 OOP 的"分层 + 面向接口编程"思想抽象为四个层次：

1. **应用层（脚本/驱动）**：负责启动、配置环境变量、选择模型/体系并调用 LAMMPS。
2. **模型层（SevenNet）**：负责网络结构（Module 组合）、数据结构（AtomGraphData）与前向/反向流程。
3. **算子层（cueq / FastEq 抽象）**：负责把 descriptor/indices/segments 映射为可执行算子对象，并提供 forward/backward。
4. **实现层（CUDA kernel + extension）**：负责具体 GPU 计算实现（mptp/cwtp/stc 等）。

3.1 UML 类图（核心对象关系图）



3.2 UML 时序图（一次 force 计算的对象协作）



3.3 面向对象职责划分（SRP）

- `RunScript`：负责环境配置与启动（不应该夹杂算子逻辑）。
- `AtomGraphSequential`：组合多个 `torch.nn.Module`，控制 forward 顺序；属于 **组合（Composition）** 模式。
- `MLIAPWrappedConvolution`：封装“图卷积”一层，内部依赖后端 `CueHelper`（适合 Strategy/Bridge）。
- `cueq_fused_scatter_channelwise_conv`：负责把 irreps/descriptor 变换成可执行 TP，多后端选择（cueq vs FastEq）。
- `SegmentedPolynomial` vs `FastEqSegmentedPolynomial`：同一抽象（“执行 segmented polynomial”）的两种实现；典型 **策略模式（Strategy）**。
- `MPTPOp/CWTPOp/STCOp`：执行层实现细节，属于“与平台相关的底层模块”，上层通过接口/绑定调用。

---

4. 功能分析（按类图视角）

4.1 功能需求（从对象协作角度）

1. **模型推理（forward）**：`Model.forward` 驱动多个 Module；每层卷积通过 `CueHelper` 间接调用 `SegmentedPolynomial` 或 `FastEqSegmentedPolynomial`。
2. **力学反向（backward）**：由 autograd 驱动，若 FastEq 路径启用，需要保证 bwd 实现正确或可靠回退。
3. **可切换后端**：通过环境变量/配置让 `CueHelper` 选择不同实现（cueq / FastEq；mptp / cwtp / naive；grouped / no-grouping）。
4. **可诊断性**：通过最小侵入式 debug（打印 descriptor/segment/shape）定位错误与性能瓶颈。

4.2 关键功能点与对应类/文件

- 后端选择：`sevenn/model_build.py`（读取 `CUEQ_USE_FASTEQ` 并向下传递）、`sevenn/nn/cue_helper.py`（use_fasteq 分支）。
- descriptor/segments 处理：`cue_helper.py` + `cuequivariance_torch.operations.tp_channel_wise.py` + `cuequivariance_torch.primitives.fasteq_segmented_polynomial.py`。

- CUDA 加速：`FastEq/fasteq/cuda/src/*` 与 `FastEq/fasteq/ops/*`。
- 性能统计：Python 侧 CUDA event 计时（mptp/cwtp）；外部 profiler（NCU）用于定位访存/warp stall。

---

5. 正确性与可替换性分析（结合类图定位改进点）

> 以“系统已可跑通”为前提，本节从 OOP 的可替换性（LSP）出发，总结 FastEq 接入后需要长期维护的正确性约束。

5.1 以 Contract 视角分析（谁保证什么）

1. `IrrepsLinear`：保证 `data["x"] -> data[key_output]` 的线性变换符合 `irreps_in/out`；前提是输入维度满足契约。
2. `MLIAPWrappedConvolution`：保证卷积层输出 `data["x"]` 的维度/语义与该层配置一致。
3. `cueq_fused_scatter_channelwise_conv`：将 descriptor 映射为可执行算子对象，并保证输出契约。
4. `SegmentedPolynomial` / `FastEqSegmentedPolynomial`：作为同一抽象的两种实现，必须在同一输入下产生语义等价输出（差异仅限性能与内部执行方式）。

5.2 LSP（里氏替换原则）落地要求

- 上层 SevenNet 不因替换后端而修改网络结构代码。
- FastEq 的 grouped/no-grouping、mptp/cwtp/naive 等内部策略变化，不得改变对外可见的输出 shape/indices/语义。
- 任何性能优化必须先通过本报告第 8 节的“契约验证护栏”。

---

6. 改进点（面向对象设计原则驱动）

6.1 SRP：拆分“构建 descriptor”与“选择执行器”

现状：`sevenn/nn/cue_helper.py` 同时承担：

- descriptor 构建与变换；
- FastEq/CUEQ 两套执行器选择；
- debug/meta 打印；
- 特殊 case（uniform_1d vs naive）处理。

改进建议：

- 引入 `DescriptorFactory`（或函数集合）专门负责 descriptor 语义构建与规范化（flatten/squeeze/split 等），保证 cueq/FastEq 使用同一套“规范 descriptor”。
- 引入 `SegmentedPolynomialExecutor` 接口（或策略类）负责选择具体执行器：cueq/SP 或 FastEqSP 或 mptp/cwtp/stc。

6.2 OCP：通过扩展增加新后端，而不是修改大量分支

现状：`use_fasteq` 作为 bool 在多个文件中传播，导致改动点多、分支多。

改进建议：

- 用策略注册表：`BackendRegistry.register("cueq", ...)`，`register("fasteq", ...)`
- 上层只持有 `IBackend` 接口引用，新增后端不需要改原逻辑（开闭原则）。

6.3 LSP：FastEq 必须“可替换”为 cueq

必须建立“替换测试”：

- 对相同 descriptor、相同 inputs，FastEq 的输出 shape/indices 约束与 cueq 完全一致；
- 对关键层做一致性断言：`assert x.shape[1] == irreps_out.dim`；
- 对数值做误差阈值检查（max_abs/max_rel）。

6.4 DIP：上层依赖抽象，避免直接依赖具体实现文件

现状：上层直接引用 `cuet.FastEqSegmentedPolynomial`（具体类）。
改进：定义抽象接口（Python 层可用 Protocol/ABC），上层依赖接口，具体实现由工厂/注入决定。

---

7. 代码改动位置清单（结合类图标注）

> 说明：本节按“类图中的对象/模块”列出可能涉及的改动位置，便于课堂汇报“改了哪、为什么改、改动影响范围”。

7.1 SevenNet 层

- `.../site-packages/sevenn/model_build.py`
  - 作用：读取 `CUEQ_USE_FASTEQ` / `use_fasteq` 并传递到 cue patch。
  - 风险：开关默认值会影响是否强制走 FastEq。
- `.../site-packages/sevenn/nn/cue_helper.py`
  - 作用：定义 `cueq_fused_scatter_channelwise_conv` 并在内部选择 `SegmentedPolynomial` 或 `FastEqSegmentedPolynomial`。
  - 风险：descriptor 规范化不一致会破坏上层输入输出契约（shape/irreps），导致隐性错配或数值问题。

7.2 cuequivariance_torch 层

- `.../cuequivariance_torch/operations/tp_channel_wise.py`、`tp_fully_connected.py`、`operations/linear.py`
  - 作用：以 `use_fasteq` 形式注入 FastEq 执行器。
  - 风险：执行器替换必须满足 LSP；否则会出现语义/shape 不一致等问题。
- `.../cuequivariance_torch/primitives/fasteq_segmented_polynomial.py`
  - 作用：FastEq meta 推导（grouped/no-grouping），以及选择 mptp/cwtp/stc/naive 等执行路径。

- 风险：meta/shape 推导必须与 cueq 等价；否则会引入 shape mismatch。

7.3 FastEq CUDA 层

- `FastEq/fasteq/cuda/src/mptp_fwd.cu`、`mptp_bwd.cu`、`cwtp_*.cu`、`stc_*.cu`
  - 作用：具体 kernel 实现与性能关键路径。
  - 风险：任何索引/布局/分组改动都可能影响数值正确性与访问越界（非法地址）。

7.4 驱动脚本层

- `lmp_single/run.sh`
  - 作用：配置环境变量并运行 LAMMPS。
  - 风险：脚本内 export 会覆盖命令行设置，造成"以为回退了其实没回退"的排障误导。