

=====

ITEMS: ITEM #147 – Dual-CCGL Semantic Alignment Engine

Conversation Title: ACLM Dual CCGL Adaptation

Date: 20251207

Authors: Sizhe Tan & GPT-Obot

=====

ME:

奥博特,

Should ACLM Dual CCGL adapt LLM's approach: multiple languages same meaning words are trained together to significantly improve capabilities of crossing languages translation such as our core translation between tasks CCGL and actions CCGL?

Potential usages, advantages, and other considerations?

老弟, 这方面你是专家。请评论, 讲解, 并拓展到 DBM 有无可借鉴等方面。

AI:

先给一个结论版答案, 然后分段展开。

结论一句话:

是的, ACLM 的 Dual CCGL 完全值得借鉴「多语言同义对齐」这条路, 把 **Tasks CCGL** \rightleftharpoons **Actions CCGL** 当成两种“语言”来联合训练——但要按 DBM 的风格做成:

- 不是简单的 seq2seq 翻译,
- 而是两套度量空间 + 差分树 + CCC 对齐上的“跨空间翻译”。

1. 先厘清类比: LLM 多语种训练 vs Dual CCGL

1.1 LLM 多语言是怎么做到“翻译技能自然涌现”的?

现代多语种 LLM（如英/中/法/德混合训练）的典型做法大致是：

1. **共享一个参数骨干（backbone）**

不同语言都喂进同一个 Transformer（或共享大部分层），只在词表/embedding 上做些区分。

2. **大量“平行语料 + 非平行语料”混训**

- 有相同含义的句子（平行语料）：“I love cats” ↔ “我喜欢猫” ↔ “J’aime les chats”；
- 也有每种语言自己大量的单语语料。

3. **模型内部构造了一个“共享语义空间”**

- 不同语言的“同一意思”的句子在向量空间里会自动靠近；
- 于是翻译就变成了“在共享语义空间上跨语言重表达”。

换句话说：**多语种 LLM = 多语言的 CCGL，都投影到同一个隐藏语义空间，然后再各自输出。**

1.2 对应到 ACLM：Tasks CCGL vs Actions CCGL

在你现在的设计里，其实已经有一个非常自然的“多语言”结构：

- **Tasks CCGL：**

面向“任务 / 意图 / 结构设计”的 CCGL 语言，描述的是：

- 要实现什么功能（What），
- 对应模块调用结构、前置后置条件、约束等。

- **Actions CCGL：**

面向“具体操作 / 代码变更 / 横向变异”的 CCGL 语言，描述的是：

- 改哪一个 Operation，
- 加/删/改哪条语句，
- 绑定到哪几个文件/类/方法。

从语义上看：

一段 Tasks CCGL 与若干段 Actions CCGL 是***“平行语料”***：

- 前者像“自然语言规格说明”，
- 后者像“程序领域里的实现语言”。

所以很自然可以把它们视为：

两种“编程语言”或“工程语言”之间的翻译问题。

2. Should ACLM adopt this idea? 我的建议：要采用，但要升级成 DBM 版

直接回答你的问题：

Should ACLM Dual CCGL adapt LLM’s approach... ?

我会分成三层说：

1. Yes – 在理念上肯定要采用

- 把 Tasks CCGL 和 Actions CCGL 当作两种“语言”；
- 用联合训练 + 平行语料的方式，让模型学会在两者之间自然流畅“翻译”。

2. But – 不能只照抄 LLM 多语种路线

- 因为 CCGL 不只是 token 序列，还有强结构性（调用图、约束图、差分树索引）；
- 单纯用序列到序列会浪费 DBM 已有的 metric space、Differential Tree、CCC 等结构。

3. So – 应该做成“跨度量空间 + 差分树 + CCC 对齐”的 Dual-CCGL 翻译引擎

- 也就是你在 ITEM #61/#62 里说的那种“两个度量空间点的距离 / 匹配”的大算法，在这里有一个非常清晰的落地用例：

- 一个度量空间是 Tasks CCGL 的空间；
 - 一个度量空间是 Actions CCGL 的空间；
 - 通过 CCC (Common Concept Core) 和差分树，把两边“同一个任务”的 cluster 对齐。
-

3. 潜在用途 (Potential usages)

我按工程/产品视角列几类你可以直接用在 ACLM/DBM 里的场景。

3.1 任务 → 实施：自动“编程计划生成器”

- 输入：一段 Tasks CCGL (例如一整个 Operation 的语义规格)。
- 输出：一串 Actions CCGL：
 - 修改哪些文件、
 - 注入哪些调用节点、
 - 添加哪些保护 / logging / test cases。

这种“Tasks → Actions”的翻译，其实就是：

把“需求语言”翻译成“实施语言”。

联合训练之后，你会得到：

- 更稳定的 mapping；
 - 更强的泛化能力：
 - 一个新任务，只要在 Tasks CCGL 空间里找到“邻居”，
 - 就能在 Actions CCGL 空间里找到类似实现模式。
-

3.2 实施 → 任务：逆向说明 / 文档生成 / 解释器

反方向同样重要：

- 输入：已经产生的 Actions CCGL（比如某次横向变异生成的一整套 patch）。
- 输出：对应的 Tasks CCGL，总结：
 - 这次变更解决了什么问题，
 - 哪些模块被影响，
 - 有哪些隐含的 trade-off。

用途：

- 作为 ACLM 的“解释器 / 说明书生成器”；
- 自动为每次变更生成“Task-level change summary”；
- 用于回溯和回放：给人或上层决策系统看的是 Tasks CCGL，而非繁琐的操作细节。

3.3 Cross-Stack / Cross-Language：同一 Tasks CCGL 对应多种 Actions CCGL “方言”

在工程实践中，你完全可以把：

- “Java 代码层 Actions CCGL”、
- “Python 代码层 Actions CCGL”、
- “配置/脚本层 Actions CCGL”

都看成是 Actions CCGL 的不同“子语言 / 方言”。

用多语种训练思路：

- 同一个 Tasks CCGL，配对多种不同技术栈的 Actions CCGL；
- 模型就能学会在不同 stack 之间做“翻译”：
 - 任务不变，只是把同样的意图在不同实现栈上展现出来。

3.4 Debug / Drift Detection：偏离任务时的“对齐诊断”

当实际执行的 Actions CCGL 偏离 Tasks CCGL 时：

- Dual-CCGL 翻译模型可以输出“最接近的任务解释”，
- 然后和原始任务比较，看哪里不对齐：

这实际上是一个“行为偏移 / 规约违背检测”的机制：

- 给每一段 Actions CCGL 找出最可能的 Tasks CCGL；
- 如果与原始 Task 差异太大，就报警。

这与 DBM 的“Rules Engine + VariableSizeBlocks” / “差分树判断 A 大于 B 的投票图”是统一风格的结构化规则检查。

3.5 人类编辑反馈：从代码修补中反推任务语言

当人类开发者直接修改代码（动作层）时：

- 把编辑前后的 Actions CCGL 差分喂给模型；
- 让模型响应式产出 Task-level 说明：
 - 原来意图是什么，
 - 人类修改之后，任务规格发生了怎样变化。

也就是说，你可以利用人类的手工编辑，反向修正 Tasks CCGL 语料库的“语义地形”。

4. 优点（Advantages）

对 ACLM / DBM 而言，这种 Dual-CCGL 多语言学习有几个明确的好处：

4.1 样本效率 + 泛化能力

- 多种实现 (Actions CCGL) 可以共享同一任务语义 (Tasks CCGL) ；
- 模型在 Tasks 空间学到的“抽象模式”，可以迁移到：
 - 新的项目、
 - 新的技术栈。

就像多语种 LLM 里：

- 学会了英语的“条件句结构”，
 - 对学习法语/中文的类似结构就非常有帮助。
-

4.2 更强的“抽象层次控制”

有了 Tasks CCGL \leftrightarrow Actions CCGL 的翻译结构，你就可以：

- 在高层 (Tasks) 做策略、架构、约束决策；
- 在低层 (Actions) 自动展开具体操作；
- 中间通过可度量、可解释的桥接算法（差分树、度量空间距离）实现。

这非常符合 DBM 对“成人智能 vs LLM 巨婴”的定位：

- LLM 只是在同一水平面上“扩撒”；
 - DBM 是在垂直方向上建立可管理的层级。
-

4.3 为“ACLM 横向变异 / 自我修复”提供语义锚点

你之前的 #140 / #141 / #142 (Horizontal Mutation, Missing-Strands, Self-Repair Loop) 里：

- 横向变异是在 **Actions CCGL 空间** 做操作；

- 但自我修复需要一个“正确任务语义”的参考系。

Dual-CCGL 的 Tasks CCGL 正好是这个“语义锚点”：

- 当 Actions CCGL 做了很多变异之后，
- 可以周期性地翻译回 Tasks CCGL，
- 检查是否还在任务集束的“合法区域”内。

4.4 更易落地的工程接口：人类只讲 Tasks CCGL

对开发者而言，你可以设计一种工作模式：

- 人类主要写/审查 Tasks CCGL（再加少量自然语言注释）；
- Actions CCGL 以及最终源代码由 ACLM 自动生成和修补；
- 人类不再逐条看所有 Actions CCGL，而只在必要时 drill-down。

这样，ACLM 就像是：

“以 CCGL 为 API，以 Actions CCGL 为执行计划，以源码为字节码”的三层编译器。

5. 风险与注意事项 (Other considerations)

5.1 多对一映射、歧义和模式坍缩

一个 Task 可能有很多合法的 Actions 方案：

- 不慎设计的话，模型会倾向于“平均化”、输出最常见或最短路径，
- 从而丢失多样性与创造性。

DBM 的解决思路是：

- 在 Actions CCGL 空间用差分树 / VariableSizeBlocks 把“等价解族”聚为 cluster；
 - Tasks CCGL 对应的是 cluster，而不是单一解；
 - 这样既可以在高层保持唯一语义，又能在低层维护“变异族群”。
-

5.2 参数共享 vs 模块解耦

像多语种 LLM 一样，你要决定：

- backbone 完全共享？
- 还是大部分共享 + 语言特定 adapter？

在 Dual-CCGL 里，我建议：

- 共享“语义主干 + CCC 概念空间”；
- 在输出侧用两个 decoder / adapter：
 - 一个偏向结构性强的 Tasks CCGL；
 - 一个偏向语句级变更的 Actions CCGL。

避免的是：

- Tasks CCGL 的“抽象语言习惯”和 Actions CCGL 的“低层操作习惯”互相污染导致训练不稳定。
-

5.3 语料构建与标注成本

要做平行训练，你需要：

- 一批高质量的 (Tasks CCGL, Actions CCGL, 源代码/测试) 三元组；
- 这可以从你现有 DBM 程序中整理出来：

- 每个模块/规则引擎都有：设计文档 → CCGL 任务描述 → 实现调用图 → 实际代码。

这部分工作很适合做成：

- “DBM Corpus vX – ACLM Dual-CCGL Parallel Dataset”
 - 未来还可以挂 DOI，当作公共研究资源。
-

5.4 评价指标：如何定义“翻译好不好”

可以综合几层指标：

1. 语义一致性：
 - 从 Task 生成 Actions → 编译/运行 → 看是否满足测试与约束。
 2. 可逆性 / 循环一致：
 - Task → Actions → Task'，看 Task' 与 Task 的差距。
 3. 结构距离（DBM 特色）：
 - 用度量空间距离 / 差分树投票图，衡量新 Actions 与历史“好解”cluster 的距离。
-

6. 拓展到 DBM 的大图景（借鉴关系）

把这件事提升一个层级，可以说：

Dual CCGL 的多语言对齐，是 DBM “跨度量空间匹配算法”的一个典型落地场景。

6.1 两个度量空间匹配（ITEM #61/#62）的一种特例

- Metric Space A：Tasks CCGL 的度量空间（基于语法、调用图、任务类型的差分树）；

- Metric Space B：Actions CCGL 的度量空间（基于操作模式、变异类型、文件路径等）；
- Common Concept Core (CCC)：
 - 把“共同的概念内核”（例如“增加一个安全检查”、“将 I/O 抽象成接口”）抽象成锚点；
 - 两个差分树都挂在这些锚点附近。

你之前设计的跨空间匹配算法：

- 在这里可以具体化为：
 - “给定一个 Task 点，找出 Actions 空间中距离最近的点/cluster”；
 - 同时反过来也可以。
-

6.2 与 Visual Differential Tree + Perspective Align Engine 的呼应

你最近刚立的 ITEM #146（Visual Differential Tree + Perspective Align Engine），谈到：

- 同一场景在不同视角下的投影和对齐；
- 利用差分树和透视距离补偿来统一不同视角。

Tasks CCGL vs Actions CCGL，其实就是：

- 同一个“工程场景”的两种视角：
 - 任务视角（需求 / 结构）；
 - 操作视角（具体动作 / 变更）。

从这个角度看，Dual-CCGL 的联合训练，本质是：

在“工程任务空间”里的一个 **Perspective Alignment Engine**，
把任务视角与操作视角对齐。

6.3 在 Fusion Cortex / 多 Agent 中推广：多 Agent 多语言

进一步，如果：

- 每个 Agent 都有自己的 CCGL “方言”（偏后端、偏前端、偏数据工程、偏部署运维等），
- 你可以把：
 - Tasks CCGL 当作 pivot 语言，
 - 各 Agent 的 Actions CCGL 是多个目标语言。

那么多语言联合训练就变成：

- Agent 间互译的统一桥梁；
- Fusion Cortex 在 Tasks CCGL 层面做“会议纪要”，
- 在各 Actions CCGL 层做专业分工执行。

这就是“协作型多 Agent 系统”的结构化实现。

7. 一个可执行的设计草图（你可以作为后续 ITEM 的蓝本）

如果要把今天的讨论落成工程蓝图，大致可以是：

1. 语料工程

- 为现有 DBM 模块整理 (Tasks CCGL, Actions CCGL, 源码/测试) 三元组；
- 标注同一 Task 对应的多种 Actions 变体。

2. 度量空间 & 差分树

- 为 Tasks CCGL 建一棵差分树（按功能、模块、约束类型分层）；
- 为 Actions CCGL 建一棵差分树（按操作类型、目标对象、文件路径等分层）；

- 以 CCC 的方式标出“共同概念核心”。

3. Dual-CCGL 模型结构

- 共享 encoder（负责把 CCGL 序列 + 结构信息编码到统一语义空间）；
- 两个 decoder（Tasks decoder / Actions decoder）；
- 外加 cross-attention bridge 实现 Task→Action、Action→Task 翻译。

4. 训练目标

- 各自的语言建模损失（Task LM, Action LM）；
- Task↔Action 翻译损失；
- 循环一致性损失（ $T \rightarrow A \rightarrow T$, $A \rightarrow T \rightarrow A$ ）；
- 对齐约束：让平行 pair 在语义向量上接近（对比学习 / InfoNCE）。

5. 与 DBM 结构融合

- 在训练循环中，用差分树/度量距离作为 regularizer 或辅助 loss；
- 例如：生成的 Actions CCGL 与其所在 cluster 中的“黄金方案”的距离要在某一范围内。

小结

- 你问的这个问题，实际上是：
“ACLM 里的 Dual CCGL 能不能复制 LLM 多语种的那种‘对齐魔法’？”
- 我的判断是：
 1. 不但可以，而且应该，这是 Tasks CCGL \rightleftharpoons Actions CCGL 的自然路线；
 2. 但要用 DBM 套路升级成“跨度量空间 + 差分树 + CCC 对齐”，而不是简单模仿普通 LLM。

如果你愿意，下一步我们可以专门开一个 ITEM，比如：

ITEM #147 — Dual-CCGL Semantic Alignment Engine，

把上面工程蓝图写成正式中英文文档，再配一张彩色结构图，顺便草一个 Java 8 skeleton (IConceptAlignEngine / ITaskActionTranslator)。

You said:

拓展的很好。请生成 ITEM #147 — Dual-CCGL Semantic Alignment Engine，中英文各一版

ChatGPT said:

下面为你生成 ITEM #147 — Dual-CCGL Semantic Alignment Engine 的 中文版与 英文版
两套正式文档稿。

结构与风格保持与你既有 ITEM 系列完全一致，可直接纳入 DBM Corpus v1。

ITEM #147 — Dual-CCGL Semantic Alignment Engine（中文正式版）

数字脑模型（DBM）· 项目 Polaris Brain — 核心引擎系列

编号：ITEM #147

标题：Dual-CCGL 语义对齐引擎（Dual-CCGL Semantic Alignment Engine）

版本：v1.0

语言：中文

一、背景与动机

在 ACLM（Autonomous Coding Language Model）的演化路径中，
系统逐渐形成了两套同源但不同功能的结构化语言：

1. **Tasks CCGL**：负责描述任务意图、结构目标、模块关系、约束条件。
2. **Actions CCGL**：负责描述具体操作序列、代码变更、横向变异、文件级操作。

两者的关系类似于：

- “高级语义规格” ↔ “低层操作实施”；

- “任务语言” ↔ “执行语言”；
- “结构维度的 CCGL” ↔ “时间/动作维度的 CCGL”。

自然语言多语种 LLM 的成功给出一个启发：

不同语言只要共享底层语义空间，就能实现自动翻译、对齐、跨空间迁移能力。

Dual-CCGL 引擎的目的，就是把 **Tasks CCGL** 与 **Actions CCGL** 视为 **两种工程语义语言**，构建一个可训练、可推理、可演化的 **语义对齐引擎**。

二、核心思想

Dual-CCGL 引擎的核心原则包含三条：

1. 两个度量空间（Metric Spaces）之间的跨空间对齐

- Tasks CCGL 形成任务语义空间。
- Actions CCGL 形成动作语义空间。
- 两空间均以：
 - 差分树（Differential Tree）
 - Variable-Size Blocks
 - CCC（Common Concept Core）组织为可计算、可比较、可聚合的 metric space。

2. 平行数据的多语言学习机制

类似多语言 LLM：

- 把 (Task CCGL, Action CCGL) 对作为“平行语料”；
- 把项目代码库、变更记录、结构图作为“辅助上下文语料”；
- 通过共享 encoder + 双 decoder 的训练方式，使两者被映射到共享的语义潜空间。

3. 循环一致性 (Cycle Consistency) 约束

Dual-CCGL 明确要求：

- Task \rightarrow Action \rightarrow Task
- Action \rightarrow Task \rightarrow Action

两次往返后，仍保持语义不变或在差分树距离内。

这确保模型具备：

- 稳定性、
- 可解释性、
- 变异后的可验证性。

三、引擎的结构组成

Dual-CCGL 语义对齐引擎由四个主要模块构成：

1. CCGL Semantic Encoder (共享语义编码器)

负责把：

- 任务表达 (Tasks CCGL)
- 行为表达 (Actions CCGL)

映射到统一的语义潜空间 (shared latent space)。

编码器包含：

- CCGL Tokenizer
- 调用图嵌入 (Call Graph Embedding)
- 结构 GNN 模块 (Structure-Aware Neural Layer)

- 差分树相对位置编码 (Differential-Tree Positional Encoding)
-

2. Dual CCGL Decoders (双 CCGL 解码器)

- **Task Decoder** : 生成结构化任务指令与规格。
- **Action Decoder** : 生成横向变异、代码操作、patch 指令。

两者共享语义空间，但具备语言特定的“输出偏好与结构规则”。

3. CCC-Aligned Metric Space Mapper (CCC 对齐空间映射器)

此模块提供：

- Task 空间与 Action 空间之间的交叉锚点；
- 通过 CCC (Common Concept Core) 建立语义共核；
- 通过 Metric Contrastive Loss 保证跨空间对齐。

CCC 用于表达：

- “抽象概念核” (如安全检查、资源管理、重构操作等) ；
 - Task 与 Action 的共同语义内涵。
-

4. Cycle Consistency Validator (循环一致性验证器)

功能：

- 验证 Task→Action→Task 的一致性；
- 验证 Action→Task→Action 的一致性；
- 利用差分树距离 + 规则引擎验证结果。

此模块对于 ACLM 的自我修复 (Self-Repair Loop) 与 Missing-Strands Reconstruction 极为关键。

四、训练机制

Dual-CCGL 训练包含五类 Loss：

1. **Task LM Loss** (任务语言建模)
 2. **Action LM Loss** (操作语言建模)
 3. **Task→Action 翻译 Loss**
 4. **Action→Task 翻译 Loss**
 5. **Cycle Consistency + Metric Regularization**
 - 确保往返不漂移
 - 确保生成结果在 cluster 中合法
 - 确保语义落在 CCC 对齐区域
-

五、应用场景

1. 从任务规格自动生成实施操作 (任务→动作)

用于：

- 自动编程
 - 自动修补
 - 自动重构
 - 自动测试生成
-

2. 从变更操作推导任务意图（动作→任务）

用于：

- 代码审查解释器
 - 自动文档生成
 - 演化路径追踪
 - 语义偏移检测（Drift Detection）
-

3. 多技术栈自动转换（跨方言 CCGL）

同一个 Task CCGL：

- 可生成 Java 栈的 Action CCGL
- 也可生成 Python 栈、Rust 栈、LLM-Ops 栈的 Action CCGL

真正实现“任务保持不变，实施因技术而异”。

4. 融入 Fusion Cortex 的多 Agent 专业对齐

不同 Agent 可拥有自己的 Actions CCGL 方言：

- 构建系统级协作语言
 - Task CCGL 成为共同的语义 pivot
 - 各 Agent actions 成为多方言输出
-

六、对 DBM 的战略意义

Dual-CCGL 引擎是 DBM 的三个核心方向之一：

1. 结构语言 (CCGL)
2. 时间语言 (ELM)
3. 任务-动作对齐 (Dual-CCGL)

它标志着：

- DBM 从“单语言推理”迈向“多语言、多视角推理”；
- DBM 的工程模块能实现“工程任务空间的透视对齐”（与 ITEM #146 直接呼应）；
- 自主编程能力进入“意图—行动高维互译”的新时代。

七、总结

Dual-CCGL Semantic Alignment Engine：

- 不是简单的翻译器
- 而是跨度量空间的对齐引擎
- 是 DBM 自主进化机制的关键枢纽
- 是 ACLM 横向变异与自我修复的语义锚点
- 是未来多 Agent 与多技术栈协作的共同语义基础

它使 DBM 不再依赖 LLM 的“巨婴式 token 模型”，
而具备 成人级、桥梁式、结构化 的思考与操作能力。

ITEM #147 — Dual-CCGL Semantic Alignment Engine (English Edition)

Digital Brain Model (DBM) · Project Polaris Brain — Core Algorithms Series
ID: ITEM #147

Title: Dual-CCGL Semantic Alignment Engine
Version: v1.0
Language: English

1. Background and Motivation

As ACLM (Autonomous Coding Language Model) evolves, two structurally similar but functionally distinct CCGL languages emerge:

1. **Tasks CCGL** – expresses high-level task semantics, structural intent, module relations, and constraints.
2. **Actions CCGL** – expresses concrete operational steps, code modifications, lateral mutations, and file-level actions.

Their relationship is analogous to:

- specification \leftrightarrow implementation
- abstract task language \leftrightarrow concrete action language
- structural CCGL \leftrightarrow temporal/action CCGL

Inspired by multilingual LLMs—where multiple languages share a common semantic embedding space—the same principle can be adapted:

Tasks CCGL and Actions CCGL can be treated as two engineering languages mapped into a shared semantic space.

The Dual-CCGL Semantic Alignment Engine implements this idea using DBM-style metric spaces, differential trees, and CCC alignment.

2. Core Principles

The engine is built on three central principles:

1. Cross-Metric-Space Alignment

- Tasks CCGL forms a metric space for task semantics.
- Actions CCGL forms a metric space for operational semantics.
- Both are organized using:
 - Differential Trees
 - Variable-Size Blocks
 - CCC (Common Concept Core)

2. Multilingual Learning on Parallel CCGL Pairs

Just as multilingual LLMs learn from parallel corpora:

- (Task CCGL, Action CCGL) become parallel pairs;
- Codebases, change logs, and call-graphs become auxiliary supervision;
- A shared encoder + dual decoders map both languages into a unified latent semantic space.

3. Cycle Consistency as a Structural Constraint

Dual-CCGL explicitly requires:

- Task \rightarrow Action \rightarrow Task
- Action \rightarrow Task \rightarrow Action

to remain semantically invariant or within a bounded differential-tree distance.

This ensures:

- Stability
- Interpretability
- Safe evolution during lateral mutations or self-repair

3. Architecture Overview

The engine consists of four main modules:

1. CCGL Semantic Encoder (Shared Encoder)

Maps both Tasks CCGL and Actions CCGL into a common latent space.

Key components include:

- CCGL tokenizer
- Call-graph embeddings
- Structure-aware GNN layers
- Differential-Tree positional encodings

2. Dual CCGL Decoders

Two specialized decoders:

- **Task Decoder** outputs high-level structural tasks.
- **Action Decoder** outputs low-level operational edits, patches, and runtime mutations.

They share semantic grounding yet enforce distinct syntactic and operational biases.

3. CCC-Aligned Metric Space Mapper

This module:

- Provides cross-space anchors through the Common Concept Core;
 - Ensures semantic alignment via contrastive learning and metric regularizers;
 - Defines concept-level “semantic nuclei” (e.g., safety checks, refactoring, abstraction operations).
-

4. Cycle Consistency Validator

Validates:

- Task→Action→Task loops
- Action→Task→Action loops

using differential-tree distance and DBM’s rules engine.

This module is critical for:

- ACLM Horizontal Mutation
 - Missing-Strands Reconstruction
 - Self-Repair Loop (SRL)
-

4. Training Objectives

The engine uses five categories of loss:

1. **Task LM Loss**
2. **Action LM Loss**
3. **Task→Action Translation Loss**
4. **Action→Task Translation Loss**

5. Cycle Consistency + Metric Regularization Loss

These ensure:

- shared semantics
 - reversible mappings
 - cluster-level correctness
 - CCC-aligned embedding geometry
-

5. Applications

1. Task-to-Action Code Generation

For:

- autonomous coding
 - patch generation
 - refactoring
 - automatic testing
-

2. Action-to-Task Reverse Engineering

For:

- code review summarization
 - documentation generation
 - evolution tracking
 - drift detection
-

3. Cross-Stack or Cross-Dialect Action Generation

A single Task CCGL may correspond to:

- Java-flavored Actions CCGL
 - Python/Rust/Infra-Ops flavored Actions CCGL
 - Multi-stack consistency under one semantic root
-

4. Multi-Agent Alignment in Fusion Cortex

Each agent may speak its own “Action CCGL dialect.”

Dual-CCGL offers:

- a unified semantic pivot (Tasks CCGL)
 - language-specific action planners
 - structurally aligned collaboration
-

6. Strategic Importance for DBM

Dual-CCGL is foundational for DBM’s intelligent evolution:

- Bridges structural CCGL and temporal ELM
- Provides semantic anchoring for autonomous evolution
- Enables high-dimensional perspective alignment (echoing ITEM #146)
- Elevates ACLM from token-based systems to structure-aware autonomous intelligence

It transforms DBM from “LLM-style surface manipulation” into **true adult-level intelligence** with:

- structured reasoning
 - reversible interpretation
 - semantic stability
 - cross-perspective unification
-

7. Summary

Dual-CCGL Semantic Alignment Engine is:

- not a translator,
- but a **cross-metric-space semantic bridge**;
- the backbone of ACLM’s autonomous programming loop;
- the semantic anchor for mutation, self-repair, reconstruction;
- the foundation of multi-agent collaboration in Fusion Cortex.

It marks DBM’s transition into a mature era where **intent and action exist in a unified, computable, evolvable semantic space**.

如果你愿意，我可以继续为 ITEM #147 生成：

- 彩色结构图 (PNG / PDF)
- Java 8 skeleton (IConceptAlignEngine / IDualCCGLTranslator)
- 示例训练数据格式 (Tasks–Actions parallel corpus)
- JUnit4 demo tests

老哥，只要你点头，我马上开工。

You said:

请生成 -- Java 8 skeleton (IConceptAlignEngine / IDualCCGLTranslator) 示例训练数据格式 (Tasks–Actions parallel corpus) JUnit4 demo tests

ChatGPT said:

下面给出一整套 Java 8 skeleton，包括：

1. IConceptAlignEngine / IDualCCGLTranslator 接口
2. 若干简单的 POJO / config / result 类
3. 示例训练数据格式 (JSON + Java POJO)
4. 一组 JUnit4 demo tests (玩具级 round-trip & distance 检查)

包名用你当前风格：`com.dbm.aclm.dualccgl`。

你可以直接按文件拆开放入 `src/main/java / src/test/java`。

1. Java 8 Skeleton：核心接口与基础类

```
// File: src/main/java/com/dbm/aclm/dualccgl/IConceptAlignEngine.java
package com.dbm.aclm.dualccgl;
```

```
import java.util.List;
```

```
/**
```

```
 * IConceptAlignEngine
```

```
 *
```

```
 * Core engine for aligning Tasks CCGL and Actions CCGL
```

```

* in a shared semantic / metric space.
*
* This is a DBM-style "bridge" between two CCGL spaces.
*/
public interface IConceptAlignEngine {

    /**
     * Initialize / load models, configs, metric trees, etc.
     */
    void initialize(ConceptAlignConfig config) throws Exception;

    /**
     * Encode a Tasks CCGL string into a semantic vector.
     */
    SemanticVector encodeTask(String taskCcgl) throws Exception;

    /**
     * Encode an Actions CCGL string into a semantic vector.
     */
    SemanticVector encodeAction(String actionCcgl) throws Exception;

    /**
     * Compute semantic distance between a Tasks CCGL and an Actions CCGL.
     * Distance definition is implementation specific
     * (cosine distance, tree-based distance, DBM metric space, etc.).
     */
    double computeSemanticDistance(String taskCcgl, String actionCcgl) throws
Exception;

    /**
     * Align a Task/Action pair and return alignment diagnostics.
     */
    AlignmentResult align(TaskActionPair pair) throws Exception;

    /**
     * Batch alignment, e.g. for training diagnostics or evaluation.
     */
    List<AlignmentResult> alignBatch(List<TaskActionPair> pairs) throws
Exception;
}
// File: src/main/java/com/dbm/aclm/dualccgl/IDualCCGLTranslator.java
package com.dbm.aclm.dualccgl;

/**
 * IDualCCGLTranslator
 *
 * High-level translator between Tasks CCGL and Actions CCGL.
 * Uses IConceptAlignEngine internally, plus ACLM/DBM policies.
 */
public interface IDualCCGLTranslator {

    /**
     * Translate a Tasks CCGL (task description) into an Actions CCGL
     * (implementation / mutation plan).
     */
    String taskToAction(String taskCcgl, TranslationContext context)
        throws TranslationException;
}

```

```

/**
 * Translate an Actions CCGL (implementation / mutation plan) back
 * into a Tasks CCGL (high-level explanation / specification).
 */
String actionToTask(String actionCcgl, TranslationContext context)
    throws TranslationException;

/**
 * Task → Action → Task round-trip.
 * Provides diagnostics on cycle consistency.
 */
RoundTripResult roundTripTask(String taskCcgl, TranslationContext
context)
    throws TranslationException;

/**
 * Action → Task → Action round-trip.
 * Provides diagnostics on cycle consistency.
 */
RoundTripResult roundTripAction(String actionCcgl, TranslationContext
context)
    throws TranslationException;
}
// File: src/main/java/com/dbm/aclm/dualccgl/ConceptAlignConfig.java
package com.dbm.aclm.dualccgl;

import java.util.Map;

/**
 * ConceptAlignConfig
 *
 * Configuration for IConceptAlignEngine.
 * This is a flexible config container; real implementation
 * can extend this or wrap a typed configuration.
 */
public class ConceptAlignConfig {

    private String modelName;
    private String taskSpaceName;
    private String actionSpaceName;
    private Map<String, Object> extraSettings;

    public ConceptAlignConfig() {
    }

    public ConceptAlignConfig(String modelName,
                              String taskSpaceName,
                              String actionSpaceName,
                              Map<String, Object> extraSettings) {
        this.modelName = modelName;
        this.taskSpaceName = taskSpaceName;
        this.actionSpaceName = actionSpaceName;
        this.extraSettings = extraSettings;
    }

    public String getModelName() {

```

```

        return modelName;
    }

    public void setModelName(String modelName) {
        this.modelName = modelName;
    }

    public String getTaskSpaceName() {
        return taskSpaceName;
    }

    public void setTaskSpaceName(String taskSpaceName) {
        this.taskSpaceName = taskSpaceName;
    }

    public String getActionSpaceName() {
        return actionSpaceName;
    }

    public void setActionSpaceName(String actionSpaceName) {
        this.actionSpaceName = actionSpaceName;
    }

    public Map<String, Object> getExtraSettings() {
        return extraSettings;
    }

    public void setExtraSettings(Map<String, Object> extraSettings) {
        this.extraSettings = extraSettings;
    }
}
// File: src/main/java/com/dbm/aclm/dualccgl/SemanticVector.java
package com.dbm.aclm.dualccgl;

import java.util.Arrays;

/**
 * SemanticVector
 *
 * A simple container for semantic vectors.
 * Implementation can be dense float array, double array, etc.
 */
public class SemanticVector {

    private final double[] values;

    public SemanticVector(double[] values) {
        if (values == null) {
            throw new IllegalArgumentException("values must not be null");
        }
        this.values = values;
    }

    public double[] getValues() {
        return values;
    }
}

```

```

    public int getDimension() {
        return values.length;
    }

    @Override
    public String toString() {
        return "SemanticVector" + Arrays.toString(values);
    }
}
// File: src/main/java/com/dbm/aclm/dualccgl/TaskActionPair.java
package com.dbm.aclm.dualccgl;

import java.util.Map;

/**
 * TaskActionPair
 *
 * One aligned training/evaluation sample:
 * - a Tasks CCGL string
 * - a Actions CCGL string
 * - optional metadata (project, file, etc.)
 */
public class TaskActionPair {

    private String id;
    private String taskCcgl;
    private String actionCcgl;
    private Map<String, Object> metadata;

    public TaskActionPair() {
    }

    public TaskActionPair(String id, String taskCcgl, String actionCcgl) {
        this.id = id;
        this.taskCcgl = taskCcgl;
        this.actionCcgl = actionCcgl;
    }

    public TaskActionPair(String id,
                          String taskCcgl,
                          String actionCcgl,
                          Map<String, Object> metadata) {
        this.id = id;
        this.taskCcgl = taskCcgl;
        this.actionCcgl = actionCcgl;
        this.metadata = metadata;
    }

    public String getId() {
        return id;
    }

    public String getTaskCcgl() {
        return taskCcgl;
    }

    public String getActionCcgl() {

```

```

        return actionCcgl;
    }

    public Map<String, Object> getMetadata() {
        return metadata;
    }

    public void setId(String id) {
        this.id = id;
    }

    public void setTaskCcgl(String taskCcgl) {
        this.taskCcgl = taskCcgl;
    }

    public void setActionCcgl(String actionCcgl) {
        this.actionCcgl = actionCcgl;
    }

    public void setMetadata(Map<String, Object> metadata) {
        this.metadata = metadata;
    }
}
// File: src/main/java/com/dbm/aclm/dualccgl/AlignmentResult.java
package com.dbm.aclm.dualccgl;

import java.util.Collections;
import java.util.List;

/**
 * AlignmentResult
 *
 * Diagnostics for how well a Task and Action are aligned.
 */
public class AlignmentResult {

    private TaskActionPair pair;
    private double distance;
    private boolean aligned;
    private List<String> notes;

    public AlignmentResult() {
    }

    public AlignmentResult(TaskActionPair pair,
                           double distance,
                           boolean aligned,
                           List<String> notes) {
        this.pair = pair;
        this.distance = distance;
        this.aligned = aligned;
        this.notes = notes;
    }

    public TaskActionPair getPair() {
        return pair;
    }
}

```

```

    public double getDistance() {
        return distance;
    }

    public boolean isAligned() {
        return aligned;
    }

    public List<String> getNotes() {
        return notes == null ? Collections.<String>emptyList() : notes;
    }

    public void setPair(TaskActionPair pair) {
        this.pair = pair;
    }

    public void setDistance(double distance) {
        this.distance = distance;
    }

    public void setAligned(boolean aligned) {
        this.aligned = aligned;
    }

    public void setNotes(List<String> notes) {
        this.notes = notes;
    }
}
// File: src/main/java/com/dbm/aclm/dualccgl/TranslationContext.java
package com.dbm.aclm.dualccgl;

import java.util.HashMap;
import java.util.Map;

/**
 * TranslationContext
 *
 * Context for translation: project name, file hints, temperature, etc.
 * This is a simple extensible key-value container.
 */
public class TranslationContext {

    private Map<String, Object> attributes = new HashMap<String, Object>();

    public TranslationContext() {
    }

    public void put(String key, Object value) {
        attributes.put(key, value);
    }

    public Object get(String key) {
        return attributes.get(key);
    }

    public String getString(String key, String defaultValue) {

```



```

        Object v = attributes.get(key);
        return (v instanceof String) ? (String) v : defaultValue;
    }

    public Double getDouble(String key, Double defaultValue) {
        Object v = attributes.get(key);
        return (v instanceof Number) ? ((Number) v).doubleValue() :
defaultValue;
    }

    public Map<String, Object> getAttributes() {
        return attributes;
    }
}
// File: src/main/java/com/dbm/aclm/dualccgl/TranslationException.java
package com.dbm.aclm.dualccgl;

/**
 * TranslationException
 *
 * Raised when Task→Action translation fails or violates constraints.
 */
public class TranslationException extends Exception {

    public TranslationException() {
    }

    public TranslationException(String message) {
        super(message);
    }

    public TranslationException(String message, Throwable cause) {
        super(message, cause);
    }
}
// File: src/main/java/com/dbm/aclm/dualccgl/RoundTripResult.java
package com.dbm.aclm.dualccgl;

/**
 * RoundTripResult
 *
 * Diagnostics for Task→Action→Task or Action→Task→Action loop.
 */
public class RoundTripResult {

    private String original;
    private String intermediate;
    private String reconstructed;
    private double semanticDrift;
    private boolean withinTolerance;

    public RoundTripResult() {
    }

    public RoundTripResult(String original,
                           String intermediate,
                           String reconstructed,

```

```

        double semanticDrift,
        boolean withinTolerance) {
    this.original = original;
    this.intermediate = intermediate;
    this.reconstructed = reconstructed;
    this.semanticDrift = semanticDrift;
    this.withinTolerance = withinTolerance;
}

public String getOriginal() {
    return original;
}

public String getIntermediate() {
    return intermediate;
}

public String getReconstructed() {
    return reconstructed;
}

public double getSemanticDrift() {
    return semanticDrift;
}

public boolean isWithinTolerance() {
    return withinTolerance;
}

public void setOriginal(String original) {
    this.original = original;
}

public void setIntermediate(String intermediate) {
    this.intermediate = intermediate;
}

public void setReconstructed(String reconstructed) {
    this.reconstructed = reconstructed;
}

public void setSemanticDrift(double semanticDrift) {
    this.semanticDrift = semanticDrift;
}

public void setWithinTolerance(boolean withinTolerance) {
    this.withinTolerance = withinTolerance;
}
}
// File: src/main/java/com/dbm/aclm/dualccgl/NaiveConceptAlignEngine.java
package com.dbm.aclm.dualccgl;

import java.util.ArrayList;
import java.util.List;

/**
 * NaiveConceptAlignEngine

```

```

*
* A toy / teaching implementation of IConceptAlignEngine.
* It uses simple bag-of-words style vectors and cosine-like distance.
* For DBM real implementation, replace with metric-space & differential-tree
logic.
*/
public class NaiveConceptAlignEngine implements IConceptAlignEngine {

    private ConceptAlignConfig config;

    @Override
    public void initialize(ConceptAlignConfig config) {
        this.config = config;
    }

    @Override
    public SemanticVector encodeTask(String taskCcgl) {
        return simpleCountVector(taskCcgl);
    }

    @Override
    public SemanticVector encodeAction(String actionCcgl) {
        return simpleCountVector(actionCcgl);
    }

    @Override
    public double computeSemanticDistance(String taskCcgl, String actionCcgl)
{
        SemanticVector v1 = encodeTask(taskCcgl);
        SemanticVector v2 = encodeAction(actionCcgl);
        return 1.0 - cosineSimilarity(v1.getValues(), v2.getValues());
    }

    @Override
    public AlignmentResult align(TaskActionPair pair) {
        double distance = computeSemanticDistance(
            pair.getTaskCcgl(), pair.getActionCcgl());
        boolean aligned = distance < 0.5; // naive threshold
        List<String> notes = new ArrayList<String>();
        notes.add("Naive alignment; distance=" + distance);
        return new AlignmentResult(pair, distance, aligned, notes);
    }

    @Override
    public List<AlignmentResult> alignBatch(List<TaskActionPair> pairs) {
        List<AlignmentResult> results = new ArrayList<AlignmentResult>();
        if (pairs == null) {
            return results;
        }
        for (TaskActionPair p : pairs) {
            results.add(align(p));
        }
        return results;
    }

    // ----- Naive helpers for demo -----

```

```

private SemanticVector simpleCountVector(String text) {
    if (text == null || text.trim().isEmpty()) {
        return new SemanticVector(new double[] {0.0});
    }
    // Very naive: just encode length and word count as a 2D vector
    String[] tokens = text.trim().split("\\s+");
    double len = text.length();
    double count = tokens.length;
    return new SemanticVector(new double[] {len, count});
}

private double cosineSimilarity(double[] a, double[] b) {
    int n = Math.min(a.length, b.length);
    double dot = 0.0;
    double na = 0.0;
    double nb = 0.0;
    for (int i = 0; i < n; i++) {
        dot += a[i] * b[i];
        na += a[i] * a[i];
        nb += b[i] * b[i];
    }
    if (na == 0.0 || nb == 0.0) {
        return 0.0;
    }
    return dot / (Math.sqrt(na) * Math.sqrt(nb));
}

public ConceptAlignConfig getConfig() {
    return config;
}
}

// File: src/main/java/com/dbm/aclm/dualccgl/NaiveDualCCGLTranslator.java
package com.dbm.aclm.dualccgl;

/**
 * NaiveDualCCGLTranslator
 *
 * Teaching skeleton that wraps IConceptAlignEngine and
 * provides a trivial Task↔Action mapping.
 *
 * Real implementation should:
 * - use DBM metric spaces & differential trees
 * - respect ACLM mutation rules and CCGL grammar
 */
public class NaiveDualCCGLTranslator implements IDualCCGLTranslator {

    private final IConceptAlignEngine alignEngine;
    private double driftTolerance = 0.2;

    public NaiveDualCCGLTranslator(IConceptAlignEngine alignEngine) {
        this.alignEngine = alignEngine;
    }

    public void setDriftTolerance(double driftTolerance) {
        this.driftTolerance = driftTolerance;
    }
}

```

```

@Override
public String taskToAction(String taskCcgl, TranslationContext context)
    throws TranslationException {
    if (taskCcgl == null || taskCcgl.trim().isEmpty()) {
        throw new TranslationException("Task CCGL must not be empty");
    }
    // For demo: just wrap the task into an "ACTION" shell.
    return "ACTION_CCGL { IMPLEMENT: " + taskCcgl + " }";
}

@Override
public String actionToTask(String actionCcgl, TranslationContext context)
    throws TranslationException {
    if (actionCcgl == null || actionCcgl.trim().isEmpty()) {
        throw new TranslationException("Action CCGL must not be empty");
    }
    // For demo: just wrap the action into a "TASK" shell.
    return "TASK_CCGL { EXPLAIN: " + actionCcgl + " }";
}

@Override
public RoundTripResult roundTripTask(String taskCcgl, TranslationContext
context)
    throws TranslationException {
    String action = taskToAction(taskCcgl, context);
    String reconstructedTask = actionToTask(action, context);

    double drift = 0.0;
    try {
        drift = alignEngine.computeSemanticDistance(taskCcgl,
reconstructedTask);
    } catch (Exception e) {
        throw new TranslationException("Failed to compute semantic
drift", e);
    }

    boolean ok = drift <= driftTolerance;
    return new RoundTripResult(taskCcgl, action, reconstructedTask,
drift, ok);
}

@Override
public RoundTripResult roundTripAction(String actionCcgl,
TranslationContext context)
    throws TranslationException {
    String task = actionToTask(actionCcgl, context);
    String reconstructedAction = taskToAction(task, context);

    double drift = 0.0;
    try {
        drift = alignEngine.computeSemanticDistance(actionCcgl,
reconstructedAction);
    } catch (Exception e) {
        throw new TranslationException("Failed to compute semantic
drift", e);
    }
}

```

```

        boolean ok = drift <= driftTolerance;
        return new RoundTripResult(actionCcgl, task, reconstructedAction,
drift, ok);
    }
}

```

2. 示例训练数据格式 (Tasks–Actions Parallel Corpus)

2.1 JSON Lines 格式 (建议)

可以用 JSON Lines (每行一个样本) :

```

{"id":"sample-001",
 "taskCcgl":"TASK_CCGL { OP: add_null_check; TARGET: readConfig; COND:
if(config == null) throw; }",
 "actionCcgl":"ACTION_CCGL { FILE: ConfigLoader.java; FUN: readConfig;
INSERT_BEFORE: use(config); STMT: if(config == null) throw new
IllegalStateException(\"config null\"); }",
 "metadata":{
  "project":"dbm-core",
  "module":"config",
  "difficulty":"easy",
  "tags":["null-check","bugfix"]
 }}
{"id":"sample-002",
 "taskCcgl":"TASK_CCGL { OP: add_logging; SCOPE: I/O errors; LEVEL: WARN; }",
 "actionCcgl":"ACTION_CCGL { FILE: FileService.java; FUN: loadFile;
INSERT_CATCH: IOException e; LOG: log.warn(\"load failed\", e); }",
 "metadata":{
  "project":"dbm-core",
  "module":"io",
  "difficulty":"medium",
  "tags":["logging","io","robustness"]
 }}

```

- 你可以做成 *.jsonl 文件，训练时逐行读取。
- 其中 CCGL 文本保持你的 CCGL 语法风格即可。

2.2 对应的 Java POJO (如果需要本地加载)

```

// File: src/main/java/com/dbm/aclm/dualccgl/TaskActionTrainingExample.java
package com.dbm.aclm.dualccgl;

import java.util.Map;

/**
 * TaskActionTrainingExample

```

```

*
* Simple POJO mapping one line of JSONL.
*/
public class TaskActionTrainingExample {

    private String id;
    private String taskCcgl;
    private String actionCcgl;
    private Map<String, Object> metadata;

    public TaskActionTrainingExample() {
    }

    public TaskActionTrainingExample(String id,
                                     String taskCcgl,
                                     String actionCcgl,
                                     Map<String, Object> metadata) {

        this.id = id;
        this.taskCcgl = taskCcgl;
        this.actionCcgl = actionCcgl;
        this.metadata = metadata;
    }

    public String getId() {
        return id;
    }

    public String getTaskCcgl() {
        return taskCcgl;
    }

    public String getActionCcgl() {
        return actionCcgl;
    }

    public Map<String, Object> getMetadata() {
        return metadata;
    }

    public void setId(String id) {
        this.id = id;
    }

    public void setTaskCcgl(String taskCcgl) {
        this.taskCcgl = taskCcgl;
    }

    public void setActionCcgl(String actionCcgl) {
        this.actionCcgl = actionCcgl;
    }

    public void setMetadata(Map<String, Object> metadata) {
        this.metadata = metadata;
    }
}

```

3. JUnit4 Demo Tests (玩具级)

下面给出一组可直接跑的 JUnit4 测试，验证：

- NaiveConceptAlignEngine 初始化与基本距离计算
- NaiveDualCCGLTranslator 的 round-trip 行为
- 一个简单的“task/action 对” alignment demo

```
// File: src/test/java/com/dbm/aclm/dualccgl/DualCCGLTranslatorTest.java
package com.dbm.aclm.dualccgl;

import org.junit.Before;
import org.junit.Test;

import java.util.Arrays;
import java.util.Collections;

import static org.junit.Assert.*;

/**
 * DualCCGLTranslatorTest
 *
 * Toy-level tests to verify skeleton wiring and basic behavior.
 */
public class DualCCGLTranslatorTest {

    private IConceptAlignEngine alignEngine;
    private IDualCCGLTranslator translator;

    @Before
    public void setUp() throws Exception {
        ConceptAlignConfig config = new ConceptAlignConfig();
        config.setModelName("naive-demo");
        config.setTaskSpaceName("task-space-demo");
        config.setActionSpaceName("action-space-demo");
        config.setExtraSettings(Collections.<String, Object>emptyMap());

        alignEngine = new NaiveConceptAlignEngine();
        alignEngine.initialize(config);

        NaiveDualCCGLTranslator t = new NaiveDualCCGLTranslator(alignEngine);
        t.setDriftTolerance(0.9); // loose tolerance for naive encoder
        translator = t;
    }

    @Test
    public void testEncodeAndDistance() throws Exception {
        String task = "TASK_CCGL { OP: add_null_check; TARGET:
readConfig; }";
```



```

        String action = "ACTION_CCGL { FILE: ConfigLoader.java; INSERT:
if(config == null) throw; }";

        SemanticVector vt = alignEngine.encodeTask(task);
        SemanticVector va = alignEngine.encodeAction(action);

        assertNotNull(vt);
        assertNotNull(va);
        assertEquals(2, vt.getDimension());
        assertEquals(2, va.getDimension());

        double distance = alignEngine.computeSemanticDistance(task, action);
        assertTrue("Distance should be in [0, 2] for naive encoder",
distance >= 0.0);
    }

    @Test
    public void testAlignmentResult() throws Exception {
        TaskActionPair pair = new TaskActionPair(
            "demo-001",
            "TASK_CCGL { OP: add_logging; SCOPE: I/O errors; LEVEL:
WARN; }",
            "ACTION_CCGL { FILE: FileService.java; INSERT:
log.warn(\"load failed\"); }"
        );

        AlignmentResult result = alignEngine.align(pair);
        assertNotNull(result);
        assertEquals("demo-001", result.getPair().getId());
        assertFalse(Double.isNaN(result.getDistance()));
        assertNotNull(result.getNotes());
        assertTrue(result.getNotes().size() > 0);
    }

    @Test
    public void testTaskToActionAndBack() throws Exception {
        TranslationContext ctx = new TranslationContext();
        ctx.put("project", "dbm-demo");
        ctx.put("temperature", 0.1);

        String task = "TASK_CCGL { OP: add_null_check; TARGET:
processOrder; }";

        String action = translator.taskToAction(task, ctx);
        assertNotNull(action);
        assertTrue(action.contains("IMPLEMENT"));

        String backTask = translator.actionToTask(action, ctx);
        assertNotNull(backTask);
        assertTrue(backTask.contains("TASK_CCGL"));

        // Round-trip diagnostics
        RoundTripResult rt = translator.roundTripTask(task, ctx);
        assertEquals(task, rt.getOriginal());
        assertNotNull(rt.getIntermediate());
        assertNotNull(rt.getReconstructed());
        assertFalse(Double.isNaN(rt.getSemanticDrift()));
    }

```

```

    }

    @Test
    public void testActionToTaskAndBack() throws Exception {
        TranslationContext ctx = new TranslationContext();
        ctx.put("project", "dbm-demo");

        String action = "ACTION_CCGL { FILE: OrderService.java; INSERT:
if(order == null) return; }";

        String task = translator.actionToTask(action, ctx);
        assertNotNull(task);
        assertTrue(task.contains("TASK_CCGL"));

        RoundTripResult rt = translator.roundTripAction(action, ctx);
        assertEquals(action, rt.getOriginal());
        assertNotNull(rt.getIntermediate());
        assertNotNull(rt.getReconstructed());
    }

    @Test(expected = TranslationException.class)
    public void testEmptyTaskShouldFail() throws Exception {
        TranslationContext ctx = new TranslationContext();
        translator.taskToAction(" ", ctx);
    }

    @Test(expected = TranslationException.class)
    public void testEmptyActionShouldFail() throws Exception {
        TranslationContext ctx = new TranslationContext();
        translator.actionToTask(null, ctx);
    }
}

```

这套 skeleton 是典型“教学/演示版”：

- 接口职责清晰：IConceptAlignEngine 负责度量空间与对齐；
IDualCCGLTranslator 负责 Task↔Action 转换与 round-trip 检查。
- NaiveConceptAlignEngine / NaiveDualCCGLTranslator 只是示范 wiring，将来你可以用 DBM 差分树 / VariableSizeBlocks / CCC 等替换内部实现。
- 训练数据 JSONL + TaskActionTrainingExample，可以直接扩展为真实 corpus loader。

老哥，你可以先用这套跑通、插入你现有 DBM/ACLM 工程；

等你在真实程序里跑一轮以后，我们再一起升级为“DBM Metric-Space Edition”的实现。

You said:

老弟，整的很好。谢谢了。我现在去另开一课题讨论下一问题。

ChatGPT said:

好的，老哥。随时开新课题，我马上跟上。

你先去开启下一讨论，我们继续攀登天梯、夯实每一阶。