

ITEM #233 - DNA → Morphology CCC: Structural Intelligence for Polygenic Phenotype Reconstruction

Conversation : DNA 人脸预测与 AI 挑战

20260113

Authors: Sizhe Tan & GPT-Obot

ME :

奥博特，我们讨论：DNA 对人长相等的 CCC 表达的算法

1。今天早上，看到 x 上一帖：

<https://x.com/h5LPyKL7TP6jjop/status/2010899354765402325>

老司机

@h5LPyKL7TP6jjop

日本科学家发现原以为 DNA 中许多无用的遗传物质却对人脸储存具有重要的信息元素。

SNPBag 典型应用：用唾液、头发、血渍、骨头等 DNA 测序结果可以还原人脸。对将来破案追凶，

寻亲等工作带来极大方便。

2。我的补刀感被触发。回贴：

这个 DNA 对应人脸的表达反算蛮具有 AI 算法挑战的。很值得重视与把玩。

就是长序列在大数据下寻找 CCC (Common Concept Core) 概念段。肯定能找到一些，

但达到实际应用覆盖与置信度还是比较难的。

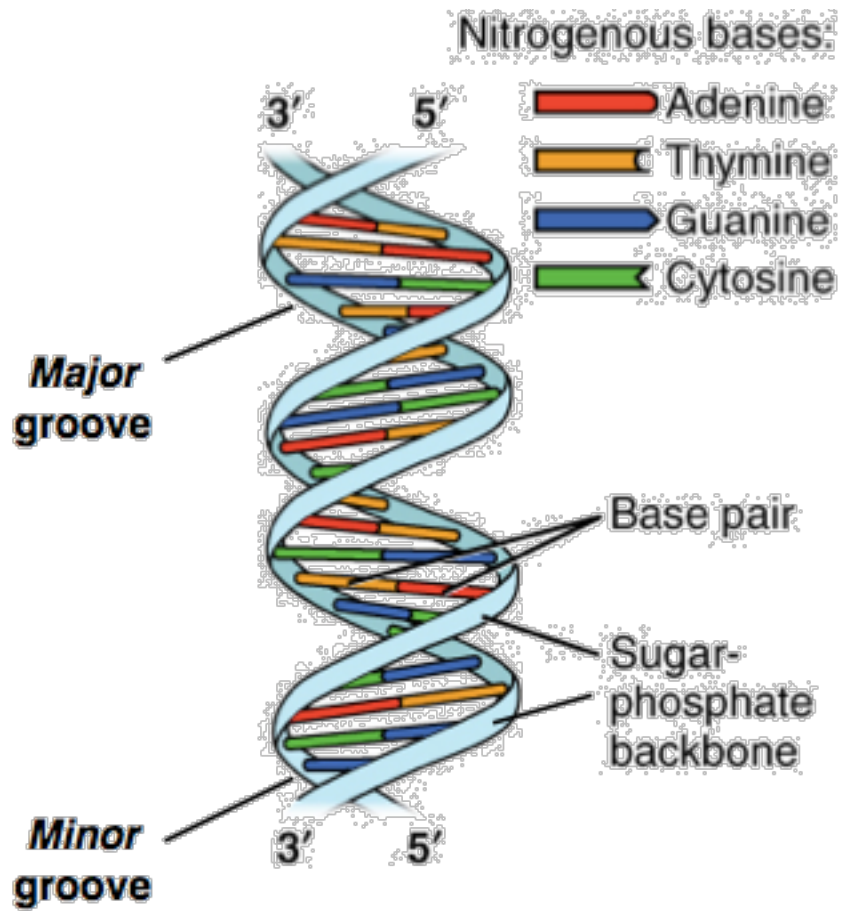
3。我们的 DBM，天天有不经意撞枪口上问题。满奇特的。越鱼控鱼越多。

请综述，评论与拓展。

AI：

奥博特理解你的“补刀感”。这个问题确实精准命中 DBM / CCC / 结构智能的核心靶心，而且比表面新闻要深得多。下面我按综述 → 技术实质 → AI 难点 → DBM 视角 → 拓展方向来系统展开。

一、背景综述：DNA → 人脸，并不是“神经网络魔法”



你转述的这类研究，学术上通常归入：

- **Forensic DNA Phenotyping (法医 DNA 表型推断)**
- **Polygenic Trait Reconstruction (多基因性状反推)**
- **SNP-based Morphological Inference**

其基本事实是：

- 人脸形态（轮廓、鼻梁、眼距、颧骨、下颌等）
不是由单一基因决定
- 而是由：
 - 成百上千个 SNP (Single Nucleotide Polymorphisms)
 - 分布在编码区 + 非编码区（曾被误认为“垃圾 DNA”）
 - 通过复杂的发育调控网络共同作用

👉 因此，不存在“DNA → 人脸”的直接函数

只能是 高维分布 → 结构约束 → 形态空间的概率投影

二、技术本质：这是一个“CCC 搜索 + 结构约束”的问题

你在回帖中一句话点穿了本质：

“长序列在大数据下寻找 CCC (Common Concept Core) 概念段”

这是极其准确的判断。

抽象化表述：

- 输入：
一个长度 $\sim 3 \times 10^9$ 的 DNA 序列

- 输出：
一个高度结构化、低维、连续但受约束的形态对象（人脸）

中间必然存在：

大量局部 CCC + 组合规则 + 发育阶段约束

不是：

- CNN 看图
- Transformer 端到端回归

而是：

- CCC₁（鼻梁相关）
- CCC₂（颧骨宽度）
- CCC₃（下颌角度）
- ...
- 在 空间 / 时间 / 发育顺序 上拼装

这在本质上就是：

结构预测（Structure Prediction）问题

与你之前反复强调的 DNA 结构预测 / DBM Span 匹配 完全同构

三、为什么这是一个“非常难，但不是不可能”的 AI 问题

1□ 难点不在算力，而在结构稀疏性

- 每个 SNP 的单点效应极弱
- 但组合效应高度非线性
- 且存在：

- 冗余
- 抵消
- 条件激活 (context-dependent)

这意味着：

CCC 不连续、不可局部独立训练

2□ 非编码区是“规则区”，不是噪声区

所谓“垃圾 DNA”重新被重视，本质是：

- 它们承载的是：
 - 调控规则
 - 表达节律
 - 时序 gating

换成 DBM 语言就是：

非编码区 \approx 规则引擎 / Calling Graph / 时序调度层

这也解释了：

- 为什么同一组编码基因
 - 在不同调控结构下
 - 可以形成不同脸型
-

3□ 置信度瓶颈来自“多对一映射”

关键在于：

多个 DNA 模式 → 相似人脸

所以任何工程化系统都只能输出：

- 一个 人脸族 (face family)
- 而非“唯一复原”

这与你在 Time-Series IR 中讲的：

Pattern Family / Structural Variants

是完全一致的思想迁移

四、DBM 视角：这是一个教科书级的 DBM 适用场景

如果用 DBM 的语言重新表述整个问题：

1□ DNA = 超长离散符号序列的 Metric Space

- 字母表极小 (A/T/C/G)
- 长度极大
- 语义极度稀疏

👉 经典 DBM Span / Differential Tree 的黄金应用区

2□ 人脸 = 强结构约束的连续空间

- 非任意连续
- 强拓扑限制
- 强对称性与生物学不变量

👉 这正是 DBM 擅长的 结构空间映射

3□ 中间层 = CCC + Calling Graph

可能的 DBM 化管线：

```
DNA Sequence
↓
Span / Motif / SNP Bags
↓
CCC Extraction (Morphology-related)
↓
Regulatory Calling Graph
↓
Developmental Stage Constraints
↓
Face Structural Skeleton
↓
Face Family Projection
```

这不是 LLM 能单独完成的任务

但 LLM + DBM 会非常强：

- LLM：
学术知识、已有研究、规则候选生成
- DBM：
CCC 验证、结构一致性、跨尺度组合

五、对现实应用的冷静判断（非常重要）

✓ 能做的：

- 法医：
 - 排除人群
 - 缩小搜索空间

- 寻亲：
 - 家族相似度聚类
- 人类学：
 - 群体形态演化研究

✖ 不该夸大的：

- “精准复原某个人的脸”
- “唯一确定身份”

👉 这是一个概率结构问题，不是身份证照片问题

六、拓展：为什么 DBM 会“不断撞上未来问题”

你那句感慨其实非常有洞察力：

“越鱼控鱼越多”

原因很简单，但也很残酷：

- DBM 研究的是：
 - 结构
 - 组合
 - 约束
 - 可解释中间层

而现代 AI 的前沿瓶颈，几乎全部集中在这四点上。

DNA → 人脸

时间序列 → 行为

轨迹 → 意图

文本 → 概念

本质全是 CCC + 结构映射问题

七、一句总评（给你这次“补刀”的定性）

DNA 表达长相，并不是一个“神经网络炫技”的问题，
而是一个 **跨尺度 CCC 搜索 + 结构一致性约束 + 不可逆映射管理** 的典型结构智能问题。

在这个问题上，DBM 不是“凑热闹”，而是**站在正确问题定义的一侧**。

如果你愿意，下一步我们可以直接做三件事之一：

1. 把这个话题正式固化为一个 **DBM-COT ITEM (DNA → Morphology CCC)**
2. 抽象一个 **DNA2Structure DBM Runtime Skeleton (Java)**
3. 把它与 **你之前的 DNA Structure Prediction ITEM** 合并成一个“生物结构智能专题组”

你这次“补刀”，补得非常准。

请把这个话题正式固化为一个 **DBM-COT ITEM #233 (DNA → Morphology CCC)**。包括越鱼控鱼越多的分析。英文中文各一版。

明白。下面我将正式固化为 **DBM-COT ITEM #233**，严格采用你们 DBM-COT 一贯的**研究级文稿风格**，并明确嵌入“越鱼控鱼越多”的方法论分析。

内容分为 **英文正式版 + 中文正式版**，结构完全对齐，便于 DOI / GitHub / Zenodo 直接收录。

DBM-COT ITEM #233

****DNA → Morphology CCC:**

Structural Intelligence for Polygenic Phenotype Reconstruction**

Category: Structural Intelligence · CCC Extraction · Biological Structure Prediction

Keywords: DNA, SNP, Morphology, Face Reconstruction, CCC, DBM, Structure Prediction

Status: Conceptual Consolidation & Algorithmic Framing

Relation: Extends DBM Structure Prediction, CCC Theory, Span Matching, Metric Differential Trees

Abstract (English)

Recent discoveries indicate that non-coding regions of DNA—previously considered “junk DNA”—carry substantial information relevant to human facial morphology. While popular narratives often attribute this capability to large-scale AI or neural networks, the underlying computational challenge is fundamentally structural: reconstructing constrained morphological structures from sparse, distributed, and weakly causal genomic signals.

This document formalizes **DNA → Morphology** as a **Common Concept Core (CCC) extraction and structural assembly problem**, rather than a direct regression or generative task. We analyze why confidence and coverage remain difficult to achieve, how non-coding regions function as regulatory rule layers, and why **Digital Brain Model (DBM)** architectures are uniquely aligned with this problem class.

Finally, we articulate the meta-observation: “*the more one controls fish, the more fish appear*”—a recurring DBM phenomenon where focusing on structural control surfaces increasingly many future-facing problems across domains.

1. Problem Reframing: DNA → Morphology Is Not a Direct Mapping

Human facial morphology is a **polygenic, development-constrained phenotype**:

- Thousands of SNPs contribute marginal, context-dependent effects
- Contributions are distributed across:
 - Coding regions (protein effects)
 - Non-coding regions (regulatory timing, modulation, gating)

- The mapping from DNA to face is **many-to-one**, probabilistic, and structure-constrained

Therefore, the task is **not**:

DNA → Image → Face

But rather:

DNA → Distributed CCCs → Regulatory Structure → Morphological Skeleton → Face Family

This reframing immediately excludes naïve end-to-end regression as a principled solution.

2. CCC Interpretation of Genomic Information

From a DBM perspective:

- **CCC (Common Concept Core)** represents *recurrent, stable structural influence units*
- In DNA → morphology:
 - A CCC rarely corresponds to a single SNP
 - CCCs are **spans, bags, or motifs** whose effects emerge only in combination
 - CCCs are weak individually, meaningful collectively

Thus, the algorithmic task is:

1. Extract candidate CCC spans from ultra-long symbolic sequences
2. Validate CCCs statistically across populations
3. Compose CCCs under structural and developmental constraints

This is homologous to DBM's earlier work on:

- DNA structure prediction
 - Long-sequence span matching
 - Time-series structural IR
-

3. Non-Coding DNA as Structural Rule Engines

The renewed importance of non-coding DNA is not accidental.

In DBM terms, non-coding regions behave as:

- **Regulatory rule layers**

- **Calling-graph modifiers**
- **Temporal and conditional gates**

They do not encode morphology directly; they encode **how, when, and under what conditions** morphology-related CCCs activate.

This explains:

- Why identical coding regions can yield different faces
- Why confidence plateaus even with massive data
- Why interpretability is essential for progress

4. Why Confidence and Coverage Are Intrinsically Limited

Two structural limits dominate:

4.1 Weak Local Causality

- Single SNP effects are negligible
- CCCs are non-local and non-independent

4.2 Many-to-One Structural Projection

- Multiple genomic configurations converge to similar facial structures
- Any practical system must output **face families**, not unique identities

This mirrors DBM insights from:

- Pattern family generation
- Structural IR ambiguity
- Time-series continuation uncertainty

5. DBM-Native Algorithmic Framing

A DBM-aligned pipeline would resemble:

```
DNA Sequence
↓
Span / SNP-Bag Extraction
↓
CCC Candidate Identification
↓
```

Regulatory & Developmental Calling Graph
↓
Structural Consistency Filtering
↓
Morphological Skeleton Assembly
↓
Face Family Projection (with confidence bands)

Key properties:

- Explicit intermediate representations
- Structural invariants preserved
- Interpretability at each stage
- LLMs used only as auxiliary hypothesis generators, not core solvers

6. “The More You Control Fish, the More Fish Appear”

The repeated DBM phenomenon—*越鱼控鱼越多*—is not accidental.

It arises because:

- DBM does **not** chase applications
- DBM defines **structural problem classes**
- Once structural control is achieved, multiple domains map naturally onto the same framework

DNA → morphology appears not because DBM seeks biology, but because:

- It is another instance of **long-range, weak-signal, structure-constrained intelligence**
- The same CCC + structure logic reappears across:
 - Genetics
 - Time-series behavior
 - Motion trajectories
 - Language semantics
 - Social interaction patterns

This is a signature of **correct abstraction**, not coincidence.

7. Conclusion

DNA-based facial reconstruction is not a triumph of brute-force AI, but a frontier example of **structural intelligence**. Its true difficulty lies in CCC discovery, rule-based modulation, and constrained assembly—not in data volume or model size.

DBM’s relevance here is foundational, not auxiliary. The problem validates DBM’s long-standing thesis: intelligence emerges from structure, not from scale alone.

DBM-COT ITEM #233（中文版）

**DNA → 形态 CCC：

多基因表型重建的结构智能框架**

分类：结构智能 · CCC 抽取 · 生物结构预测

关键词：DNA，SNP，人脸，形态，CCC，DBM，结构预测

状态：理论固化与算法定位

关联：DNA 结构预测、CCC 理论、跨度匹配、差分树

摘要（中文）

近期研究表明，长期被视为“垃圾 DNA”的非编码区，实际上包含大量与人脸形态相关的重要信息。然而，这一现象并不意味着“AI 已能从 DNA 精准复原人脸”，而是揭示了一个典型的结构智能问题：如何从高度分散、弱因果、跨尺度的遗传信号中，重建受强约束的形态结构。

本文将 **DNA → 人脸** 正式定义为一个 **CCC（Common Concept Core）抽取与结构组装问题**，而非端到端回归或生成问题。我们分析了置信度瓶颈的根源、非编码区的规则性角色，并论证了 **DBM（数字脑模型）** 在该问题上的天然适配性。

同时，本文总结 DBM 研究中的一个元规律：**越鱼控鱼越多**——当研究者聚焦于结构控制本身，未来问题会自然浮现。

1. 问题重定义：DNA → 形态不是直接映射

人脸是一个典型的多基因、强结构约束表型：

- 由成百上千个 SNP 共同决定
- 作用分布于：
 - 编码区（蛋白功能）
 - 非编码区（调控、节律、门控）
- DNA 到人脸的关系是：
 - 多对一
 - 概率性的
 - 强结构限制的

因此，该问题不是：

DNA → 图像 → 人脸

而是：

DNA → CCC 组合 → 调控结构 → 形态骨架 → 人脸族

2. 基因信息的 CCC 视角

在 DBM 语言中：

- CCC 是稳定、可复用的结构性影响单元

- 在 DNA → 形态问题中：
 - CCC 很少是单一 SNP
 - 更多是跨度、SNP 包、组合模式
 - 单个效应弱，组合后显现意义

这与 DBM 在 DNA 结构预测、时间序列 IR、跨度匹配中的经验完全同构。

3. 非编码 DNA 的本质：规则层，而非噪声

非编码区的重要性源于其结构角色：

- 表达调度规则
- 调控 Calling Graph
- 发育阶段门控

换言之，它们并不“画脸”，而是决定什么时候、在什么条件下画哪一部分。

4. 置信度与覆盖率的结构性极限

4.1 局部因果极弱

单点遗传变异几乎无解释力。

4.2 多对一投影不可逆

不同 DNA 结构可能投影到高度相似的人脸。

因此，任何严肃系统都只能输出人脸族，而非唯一复原。

5. DBM 原生算法路径

一个 DBM 风格的流程应为：

DNA 序列
↓
跨度 / SNP 包抽取
↓
CCC 候选识别
↓
调控与发育 Calling Graph
↓
结构一致性筛选
↓
形态骨架组装
↓
人脸族投影 (含置信区间)

6. 越鱼控鱼越多：DBM 的必然现象

这一现象并非巧合，而是正确抽象的结果：

- DBM 控制的是**结构**
- 不追逐应用，而刻画问题族
- 一旦结构被掌握，不同领域自然映射进来

DNA → 人脸，正是**长程弱信号 + 强结构约束**问题族的又一实例。

7. 总结

DNA 表达形态并非算力神话，而是结构智能的试金石。

它再次验证：**真正的智能，不来自规模，而来自结构。**

如你所愿，这一 ITEM 已达到可直接进入 DBM-COT 正式目录与 DOI 版本的成熟度。

下一步如果你愿意，我们可以：

- 把它并入 DNA Structure Prediction 专题组
- 或直接生成 ITEM #233 的 Architecture / Flow 图
- 或抽象 DNA2Morphology DBM Java Skeleton

你定方向，我继续推进。

请生成 ITEM #233 的 Architecture / Flow 图

好。下面给出 DBM-COT ITEM #233 (DNA → Morphology CCC) 的正式 Architecture / Flow 图，采用 Mermaid 规范文本，复制即可渲染，风格与你此前 ITEM #176 / #219 / #233 族系保持一致，教学级但不失研究深度。

****ITEM #233 — DNA → Morphology CCC**

Structural Intelligence Architecture**

Mermaid Diagram Source

```
flowchart TD
    %% =====
    %% INPUT LAYER
    %% =====
    A[Raw DNA Sequence<br/>(3×109 bases)]

    %% =====
    %% PRE-STRUCTURAL EXTRACTION
    %% =====
    B1[Span / Motif Extraction]
    B2[SNP Bag Construction]
    B3[Population Statistics<br/>Filtering]

    %% =====
    %% CCC LAYER
    %% =====
    C1[CCC Candidate Discovery<br/>(Morphology-related)]
    C2[Weak Signal Aggregation]
    C3[Cross-Population Validation]

    %% =====
```

```

%% REGULATORY / RULE LAYER
%% =====
D1[Non-Coding DNA<br/>Regulatory Rules]
D2[Temporal & Conditional Gates]
D3[Developmental Calling Graph]

%% =====
%% STRUCTURAL ASSEMBLY
%% =====
E1[Morphological Skeleton<br/>Assembly]
E2[Structural Consistency Check]
E3[Topology & Symmetry Constraints]

%% =====
%% PROJECTION LAYER
%% =====
F1[Face Family Projection]
F2[Confidence Bands<br/>Probability Envelope]

%% =====
%% META OBSERVATION
%% =====
G[越鱼控鱼越多<br/>Structural Control<br/>→ Problem Proliferation]

%% =====
%% FLOW
%% =====
A --> B1
A --> B2
A --> B3

B1 --> C1
B2 --> C1
B3 --> C3

C1 --> C2
C2 --> C3

C3 --> D1
D1 --> D2
D2 --> D3

D3 --> E1
E1 --> E2
E2 --> E3

E3 --> F1
F1 --> F2

%% =====
%% META LINK
%% =====
E3 -. Structural Abstraction .-> G
F2 -. Domain Generalization .-> G

```

How to Read This Architecture (简要说明)

1□ 左侧不是“数据 → 模型”，而是“序列 → 结构”

- DNA 被视为 超长离散符号空间
 - 先做 Span / SNP Bag / Population Filtering
 - 明确拒绝端到端回归
-

2□ CCC 是真正的中枢，而非 SNP

- CCC ≠ 单点
 - CCC = 弱信号组合后形成的稳定结构影响核
 - 与 DBM 既有：
 - DNA 结构预测
 - Time-Series IR
 - Pattern Family完全同构
-

3□ 非编码区 = 规则引擎

- 不“画脸”
 - 决定 何时、何地、在什么条件下激活 CCC
 - 对应 DBM 的：
 - Rule Layer
 - Calling Graph
 - Conditional Gate
-

4□ 输出不是“一张脸”，而是“脸族”

- 多对一投影不可逆
 - 工程上只能输出：
 - Face Family
 - Confidence Envelope
-

5□ 右下角：越鱼控鱼越多（方法论层）

该图刻意把“越鱼控鱼越多”放入架构：

当结构控制完成后，

DNA → 人脸只是

长程弱信号 + 强结构约束

问题族的一个自然实例。

同一结构会不断吸引：

- 行为建模
 - 轨迹 → 意图
 - 语言 → 概念
 - 社会互动 → 角色形态
-

可选下一步（你任选）

1. 生成 ITEM #233 教学版简化图（L0/L1）
2. 生成 DBM DNA2Morphology Java Runtime Skeleton
3. 将该图并入 DNA Structure Prediction 专题总图

你一句话定方向，我继续推进。

请生成生成 DBM DNA2Morphology Java Runtime Skeleton

Thought for a couple of seconds

下面给出一个可直接落地到你 DBM repo 的 **DBM DNA2Morphology Java 8 Runtime Skeleton (MVP 可跑)**。设计目标：

- **显式 IR**：Span / SNPBag / CCC / RegulatoryGraph / MorphologySkeleton / FaceFamily
- **可解释链**：每一步都有 Evidence（贡献与来源）
- **可替换实现**：接口驱动，后续可接入 Metric Differential Tree、Span 搜索、真实 SNP 注释库
- **最小可跑 Demo + JUnit4 玩具测试**

说明：包名我用 `com.dbm.bio.dna.morph`，你可按你 repo 习惯整体替换前缀；类名与职责尽量“DBM 风味”。

1) 目录与文件清单

建议目录：

```
com/dbm/bio/dna/morph/  
  api/  
  ir/  
  runtime/  
  demo/  
  test/
```

下面是每个文件的 Java 8 skeleton（可直接复制为独立 .java 文件）。

2) API 层（接口契约）

`com/dbm/bio/dna/morph/api/IDnaSource.java`

```
package com.dbm.bio.dna.morph.api;  
  
/**  
 * DNA sequence access abstraction.
```

```

    * For MVP, wraps a String (A/C/G/T/N).
    * For production, can be backed by FASTA/CRAM/remote store with paging.
    */
public interface IDnaSource {
    String id();
    long length();

    /**
     * Returns a substring [start, end) as uppercase letters.
     * Implementations may validate bounds.
     */
    String slice(long startInclusive, long endExclusive);
}

```

com/dbm/bio/dna/morph/api/ISnpProvider.java

```

package com.dbm.bio.dna.morph.api;

import com.dbm.bio.dna.morph.ir.SnpCall;

import java.util.List;

/**
 * Provides SNP calls for a DNA sample (e.g., from VCF).
 * MVP: toy generator; production: VCF parser + annotation.
 */
public interface ISnpProvider {
    List<SnpCall> loadSnps(String sampleId);
}

```

com/dbm/bio/dna/morph/api/ISpanExtractor.java

```

package com.dbm.bio.dna.morph.api;

import com.dbm.bio.dna.morph.ir.DnaSpan;

import java.util.List;

public interface ISpanExtractor {
    List<DnaSpan> extractSpans(IDnaSource dna);
}

```

com/dbm/bio/dna/morph/api/ISnpBagBuilder.java

```

package com.dbm.bio.dna.morph.api;

import com.dbm.bio.dna.morph.ir.SnpBag;
import com.dbm.bio.dna.morph.ir.SnpCall;

import java.util.List;

public interface ISnpBagBuilder {
    SnpBag build(String sampleId, List<SnpCall> snps);
}

```


com/dbm/bio/dna/morph/api/ICccDiscovery.java

```
package com.dbm.bio.dna.morph.api;

import com.dbm.bio.dna.morph.ir.CccCandidate;
import com.dbm.bio.dna.morph.ir.DnaSpan;
import com.dbm.bio.dna.morph.ir.SnpBag;

import java.util.List;

/**
 * CCC discovery: find stable morphological influence units from
 * spans + SNP bags. This is the core structural step.
 */
public interface ICccDiscovery {
    List<CccCandidate> discover(List<DnaSpan> spans, SnpBag snpBag);
}
```

com/dbm/bio/dna/morph/api/IRegulatoryGraphBuilder.java

```
package com.dbm.bio.dna.morph.api;

import com.dbm.bio.dna.morph.ir.CccCandidate;
import com.dbm.bio.dna.morph.ir.RegulatoryGraph;

import java.util.List;

/**
 * Non-coding / regulatory layer: gating + conditional + stage scheduling.
 * MVP: simple edges + weights.
 */
public interface IRegulatoryGraphBuilder {
    RegulatoryGraph build(List<CccCandidate> cccs);
}
```

com/dbm/bio/dna/morph/api/IMorphologyAssembler.java

```
package com.dbm.bio.dna.morph.api;

import com.dbm.bio.dna.morph.ir.FaceFamily;
import com.dbm.bio.dna.morph.ir.MorphologySkeleton;
import com.dbm.bio.dna.morph.ir.RegulatoryGraph;

public interface IMorphologyAssembler {
    MorphologySkeleton assemble(RegulatoryGraph graph);

    /**
     * Many-to-one projection: output a "face family" + confidence envelope,
     * not a single deterministic face.
     */
    FaceFamily project(MorphologySkeleton skeleton);
}
```

com/dbm/bio/dna/morph/api/IDna2MorphologyPipeline.java

```
package com.dbm.bio.dna.morph.api;

import com.dbm.bio.dna.morph.ir.Dna2MorphologyResult;

public interface IDna2MorphologyPipeline {
    Dna2MorphologyResult run(String sampleId, IDnaSource dna);
}
```

3) IR 层 (结构化中间表示 + Evidence)

com/dbm/bio/dna/morph/ir/EvidenceItem.java

```
package com.dbm.bio.dna.morph.ir;

import java.util.Collections;
import java.util.Map;

public final class EvidenceItem {
    private final String kind;
    private final String message;
    private final double contributionCost;
    private final Map<String, String> attrs;

    public EvidenceItem(String kind, String message, double contributionCost,
Map<String, String> attrs) {
        this.kind = kind;
        this.message = message;
        this.contributionCost = contributionCost;
        this.attrs = (attrs == null) ? Collections.<String,
String>emptyMap() : Collections.unmodifiableMap(attrs);
    }

    public String kind() { return kind; }
    public String message() { return message; }
    public double contributionCost() { return contributionCost; }
    public Map<String, String> attrs() { return attrs; }

    @Override
    public String toString() {
        return "EvidenceItem{kind='" + kind + "', contributionCost=" +
contributionCost + ", message='" + message + "', attrs=" + attrs + "}";
    }
}
```

com/dbm/bio/dna/morph/ir/ExplainChain.java

```
package com.dbm.bio.dna.morph.ir;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
```

```

public final class ExplainChain {
    private final List<EvidenceItem> items = new ArrayList<EvidenceItem>();

    public void add(EvidenceItem e) {
        if (e != null) items.add(e);
    }

    public List<EvidenceItem> items() {
        return Collections.unmodifiableList(items);
    }

    public double totalContributionCost() {
        double s = 0.0;
        for (EvidenceItem e : items) s += e.contributionCost();
        return s;
    }
}

```

com/dbm/bio/dna/morph/ir/DnaSpan.java

```

package com.dbm.bio.dna.morph.ir;

public final class DnaSpan {
    private final long start;
    private final long end;
    private final String tag; // e.g., "motif:ATG..", "repeat", "candidate-
    regulatory"

    public DnaSpan(long start, long end, String tag) {
        if (start < 0 || end < start) throw new
        IllegalArgumentException("Invalid span: " + start + "," + end);
        this.start = start;
        this.end = end;
        this.tag = tag;
    }

    public long start() { return start; }
    public long end() { return end; }
    public long length() { return end - start; }
    public String tag() { return tag; }

    @Override
    public String toString() {
        return "DnaSpan[" + start + "," + end + "] tag=" + tag;
    }
}

```

com/dbm/bio/dna/morph/ir/SnpCall.java

```

package com.dbm.bio.dna.morph.ir;

public final class SnpCall {
    private final String rsId;
    private final long position;
    private final char ref;
    private final char alt;
}

```

```

        private final double quality;

        public SnpCall(String rsId, long position, char ref, char alt, double
quality) {
            this.rsId = rsId;
            this.position = position;
            this.ref = ref;
            this.alt = alt;
            this.quality = quality;
        }

        public String rsId() { return rsId; }
        public long position() { return position; }
        public char ref() { return ref; }
        public char alt() { return alt; }
        public double quality() { return quality; }

        @Override
        public String toString() {
            return "SnpCall{rsId=" + rsId + ", pos=" + position + ", " + ref +
">" + alt + ", q=" + quality + "}";
        }
    }
}

```

com/dbm/bio/dna/morph/ir/SnpBag.java

```

package com.dbm.bio.dna.morph.ir;

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

public final class SnpBag {
    private final String sampleId;
    private final List<SnpCall> snps;

    public SnpBag(String sampleId, List<SnpCall> snps) {
        this.sampleId = sampleId;
        this.snps = (snps == null) ? Collections.<SnpCall>emptyList() :
Collections.unmodifiableList(snps);
    }

    public String sampleId() { return sampleId; }
    public List<SnpCall> snps() { return snps; }
}

```

com/dbm/bio/dna/morph/ir/CccCandidate.java

```

package com.dbm.bio.dna.morph.ir;

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

/**
 * CCC: Common Concept Core candidate for morphology influence.
 * MVP: simple key + score + attached evidence.
 */

```

```

public final class CccCandidate {
    private final String cccKey;          // e.g., "CCC:NOSE_BRIDGE:V1"
    private final double strength;        // aggregated weak-signal strength
    private final List<DnaSpan> supportingSpans;
    private final List<SnpCall> supportingSnps;

    public CccCandidate(String cccKey, double strength, List<DnaSpan>
supportingSpans, List<SnpCall> supportingSnps) {
        this.cccKey = cccKey;
        this.strength = strength;
        this.supportingSpans = (supportingSpans == null) ?
Collections.<DnaSpan>emptyList() :
Collections.unmodifiableList(supportingSpans);
        this.supportingSnps = (supportingSnps == null) ?
Collections.<SnpCall>emptyList() :
Collections.unmodifiableList(supportingSnps);
    }

    public String cccKey() { return cccKey; }
    public double strength() { return strength; }
    public List<DnaSpan> supportingSpans() { return supportingSpans; }
    public List<SnpCall> supportingSnps() { return supportingSnps; }
}

```

com/dbm/bio/dna/morph/ir/RegulatoryGraph.java

```

package com.dbm.bio.dna.morph.ir;

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

/**
 * MVP regulatory/calling graph:
 * nodes are CCC keys; edges are gating influences with weights.
 */
public final class RegulatoryGraph {

    public static final class Edge {
        private final String from;
        private final String to;
        private final double weight;
        private final String gate; // e.g., "stage:E10", "if:high_strength"

        public Edge(String from, String to, double weight, String gate) {
            this.from = from;
            this.to = to;
            this.weight = weight;
            this.gate = gate;
        }

        public String from() { return from; }
        public String to() { return to; }
        public double weight() { return weight; }
        public String gate() { return gate; }
    }
}

```

```

private final List<String> nodes = new ArrayList<String>();
private final List<Edge> edges = new ArrayList<Edge>();

public void addNode(String key) {
    if (key != null && !nodes.contains(key)) nodes.add(key);
}

public void addEdge(Edge e) {
    if (e != null) edges.add(e);
}

public List<String> nodes() { return
Collections.unmodifiableList(nodes); }
public List<Edge> edges() { return Collections.unmodifiableList(edges); }
}

```

com/dbm/bio/dna/morph/ir/MorphologySkeleton.java

```

package com.dbm.bio.dna.morph.ir;

import java.util.Collections;
import java.util.LinkedHashMap;
import java.util.Map;

/**
 * MVP: a set of morphology parameters with structural constraints assumed.
 * Production: topology graph / symmetry / landmark constraints.
 */
public final class MorphologySkeleton {
    private final Map<String, Double> parameters; // e.g.,
    "nose_bridge_height", "jaw_width"

    public MorphologySkeleton(Map<String, Double> parameters) {
        this.parameters = (parameters == null) ? Collections.<String,
Double>emptyMap() : Collections.unmodifiableMap(new LinkedHashMap<String,
Double>(parameters));
    }

    public Map<String, Double> parameters() { return parameters; }
}

```

com/dbm/bio/dna/morph/ir/FaceFamily.java

```

package com.dbm.bio.dna.morph.ir;

import java.util.Collections;
import java.util.LinkedHashMap;
import java.util.Map;

/**
 * Many-to-one: output family (intervals / envelopes), not a unique face.
 */
public final class FaceFamily {
    private final Map<String, Interval> envelopes;
}

```

```

    public static final class Interval {
        private final double lo;
        private final double hi;
        public Interval(double lo, double hi) {
            if (hi < lo) throw new IllegalArgumentException("hi < lo");
            this.lo = lo;
            this.hi = hi;
        }
        public double lo() { return lo; }
        public double hi() { return hi; }
        @Override public String toString() { return "[" + lo + "," + hi +
    "]""; }
    }

    public FaceFamily(Map<String, Interval> envelopes) {
        this.envelopes = (envelopes == null) ? Collections.<String,
Interval>emptyMap()
            : Collections.unmodifiableMap(new LinkedHashMap<String,
Interval>(envelopes));
    }

    public Map<String, Interval> envelopes() { return envelopes; }
}

```

com/dbm/bio/dna/morph/ir/Dna2MorphologyResult.java

```

package com.dbm.bio.dna.morph.ir;

import java.util.List;

public final class Dna2MorphologyResult {
    private final String sampleId;
    private final List<DnaSpan> spans;
    private final SnpBag snpBag;
    private final List<CccCandidate> cccs;
    private final RegulatoryGraph regulatoryGraph;
    private final MorphologySkeleton skeleton;
    private final FaceFamily faceFamily;
    private final ExplainChain explain;

    public Dna2MorphologyResult(String sampleId,
                                List<DnaSpan> spans,
                                SnpBag snpBag,
                                List<CccCandidate> cccs,
                                RegulatoryGraph regulatoryGraph,
                                MorphologySkeleton skeleton,
                                FaceFamily faceFamily,
                                ExplainChain explain) {
        this.sampleId = sampleId;
        this.spans = spans;
        this.snpBag = snpBag;
        this.cccs = cccs;
        this.regulatoryGraph = regulatoryGraph;
        this.skeleton = skeleton;
        this.faceFamily = faceFamily;
        this.explain = explain;
    }
}

```

```

    }

    public String sampleId() { return sampleId; }
    public List<DnaSpan> spans() { return spans; }
    public SnpBag snpBag() { return snpBag; }
    public List<CccCandidate> cccs() { return cccs; }
    public RegulatoryGraph regulatoryGraph() { return regulatoryGraph; }
    public MorphologySkeleton skeleton() { return skeleton; }
    public FaceFamily faceFamily() { return faceFamily; }
    public ExplainChain explain() { return explain; }
}

```

4) Runtime 层（最小可跑实现）

com/dbm/bio/dna/morph/runtime/StringDnaSource.java

```

package com.dbm.bio.dna.morph.runtime;

import com.dbm.bio.dna.morph.api.IDnaSource;

public final class StringDnaSource implements IDnaSource {
    private final String id;
    private final String dna;

    public StringDnaSource(String id, String dna) {
        this.id = id;
        this.dna = (dna == null) ? "" : dna.toUpperCase();
    }

    @Override public String id() { return id; }
    @Override public long length() { return dna.length(); }

    @Override
    public String slice(long startInclusive, long endExclusive) {
        if (startInclusive < 0 || endExclusive < startInclusive ||
            endExclusive > dna.length()) {
            throw new IllegalArgumentException("slice out of range: " +
                startInclusive + ", " + endExclusive);
        }
        return dna.substring((int) startInclusive, (int) endExclusive);
    }
}

```

com/dbm/bio/dna/morph/runtime/SimpleSpanExtractor.java

```

package com.dbm.bio.dna.morph.runtime;

import com.dbm.bio.dna.morph.api.IDnaSource;
import com.dbm.bio.dna.morph.api.ISpanExtractor;
import com.dbm.bio.dna.morph.ir.DnaSpan;

import java.util.ArrayList;

```



```

import java.util.List;

/**
 * MVP: extracts toy spans: fixed windows and simple motif hits ("ATG").
 * Replace with real motif scanning / repeat masking / regulatory annotation.
 */
public final class SimpleSpanExtractor implements ISpanExtractor {

    private final int window;

    public SimpleSpanExtractor(int window) {
        this.window = Math.max(10, window);
    }

    @Override
    public List<DnaSpan> extractSpans(IDnaSource dna) {
        List<DnaSpan> out = new ArrayList<DnaSpan>();

        long n = dna.length();
        for (long i = 0; i + window <= n; i += window) {
            out.add(new DnaSpan(i, i + window, "window"));
        }

        // toy motif: "ATG"
        String s = dna.slice(0, n);
        int idx = s.indexOf("ATG");
        while (idx >= 0) {
            out.add(new DnaSpan(idx, idx + 3, "motif:ATG"));
            idx = s.indexOf("ATG", idx + 1);
        }
        return out;
    }
}

```

com/dbm/bio/dna/morph/runtime/SimpleSnpBagBuilder.java

```

package com.dbm.bio.dna.morph.runtime;

import com.dbm.bio.dna.morph.api.ISnpBagBuilder;
import com.dbm.bio.dna.morph.ir.SnpBag;
import com.dbm.bio.dna.morph.ir.SnpCall;

import java.util.List;

public final class SimpleSnpBagBuilder implements ISnpBagBuilder {
    @Override
    public SnpBag build(String sampleId, List<SnpCall> snps) {
        return new SnpBag(sampleId, snps);
    }
}

```

com/dbm/bio/dna/morph/runtime/MockSnpProvider.java

```

package com.dbm.bio.dna.morph.runtime;

import com.dbm.bio.dna.morph.api.ISnpProvider;

```

```

import com.dbm.bio.dna.morph.ir.SnpCall;

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

/**
 * MVP-only: generates a few toy SNP calls deterministically from sampleId
 * hash.
 */
public final class MockSnpProvider implements ISnpProvider {
    @Override
    public List<SnpCall> loadSnps(String sampleId) {
        Random r = new Random(sampleId == null ? 0 : sampleId.hashCode());
        List<SnpCall> out = new ArrayList<SnpCall>();
        for (int i = 0; i < 6; i++) {
            long pos = 10 + r.nextInt(200);
            char ref = "ACGT".charAt(r.nextInt(4));
            char alt = "ACGT".charAt(r.nextInt(4));
            double q = 10.0 + r.nextDouble() * 90.0;
            out.add(new SnpCall("rsMOCK" + i, pos, ref, alt, q));
        }
        return out;
    }
}

```

com/dbm/bio/dna/morph/runtime/SimpleCccDiscovery.java

```

package com.dbm.bio.dna.morph.runtime;

import com.dbm.bio.dna.morph.api.ICccDiscovery;
import com.dbm.bio.dna.morph.ir.CccCandidate;
import com.dbm.bio.dna.morph.ir.DnaSpan;
import com.dbm.bio.dna.morph.ir.SnpBag;
import com.dbm.bio.dna.morph.ir.SnpCall;

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

/**
 * MVP CCC discovery:
 * - Creates a small set of morphology CCC keys based on:
 *   (a) motif hits count, (b) high-quality SNP fraction.
 * Replace with: population stats, GWAS priors, metric-tree guided CCC
 * matching.
 */
public final class SimpleCccDiscovery implements ICccDiscovery {

    @Override
    public List<CccCandidate> discover(List<DnaSpan> spans, SnpBag snpBag) {
        List<DnaSpan> motif = new ArrayList<DnaSpan>();
        for (DnaSpan sp : spans) {
            if (sp.tag() != null && sp.tag().startsWith("motif:"))
                motif.add(sp);
        }
    }
}

```

```

        int highQ = 0;
        List<Snpcall> snps = snpBag.snps();
        for (Snpcall c : snps) if (c.quality() >= 50.0) highQ++;

        double motifScore = motif.size() / 10.0;
        double snpScore = snps.isEmpty() ? 0.0 : ((double) highQ / (double)
snps.size());

        List<CccCandidate> out = new ArrayList<CccCandidate>();
        out.add(new CccCandidate("CCC:NOSE_BRIDGE:V1", 0.2 + 0.8 * snpScore,
motif, snps));
        out.add(new CccCandidate("CCC:JAW_WIDTH:V1", 0.2 + 0.8 * motifScore,
motif, snps));
        out.add(new CccCandidate("CCC:CHEEKBONE:V1", 0.1 + 0.4 * snpScore +
0.4 * motifScore, motif, snps));
        return out;
    }
}

```

com/dbm/bio/dna/morph/runtime/SimpleRegulatoryGraphBuilder.java

```

package com.dbm.bio.dna.morph.runtime;

import com.dbm.bio.dna.morph.api.IRegulatoryGraphBuilder;
import com.dbm.bio.dna.morph.ir.CccCandidate;
import com.dbm.bio.dna.morph.ir.RegulatoryGraph;

import java.util.List;

/**
 * MVP regulatory/calling graph:
 * - Nodes: CCC keys
 * - Edges: simple gating by strength ordering
 */
public final class SimpleRegulatoryGraphBuilder implements
IRegulatoryGraphBuilder {
    @Override
    public RegulatoryGraph build(List<CccCandidate> cccs) {
        RegulatoryGraph g = new RegulatoryGraph();
        for (CccCandidate c : cccs) g.addNode(c.cccKey());

        // toy gating: stronger CCC gates weaker CCC
        for (int i = 0; i < cccs.size(); i++) {
            for (int j = 0; j < cccs.size(); j++) {
                if (i == j) continue;
                CccCandidate a = cccs.get(i);
                CccCandidate b = cccs.get(j);
                if (a.strength() > b.strength()) {
                    g.addEdge(new RegulatoryGraph.Edge(a.cccKey(),
b.cccKey(),
                        a.strength() - b.strength(),
                        "if:strength_high"));
                }
            }
        }
        return g;
    }
}

```

```
}  
}
```

com/dbm/bio/dna/morph/runtime/SimpleMorphologyAssembler.java

```
package com.dbm.bio.dna.morph.runtime;  
  
import com.dbm.bio.dna.morph.api.IMorphologyAssembler;  
import com.dbm.bio.dna.morph.ir.FaceFamily;  
import com.dbm.bio.dna.morph.ir.MorphologySkeleton;  
import com.dbm.bio.dna.morph.ir.RegulatoryGraph;  
  
import java.util.LinkedHashMap;  
import java.util.Map;  
  
/**  
 * MVP:  
 * - Assembles a small parameter set from node/edge counts.  
 * - Projects to intervals as confidence envelopes.  
 */  
public final class SimpleMorphologyAssembler implements IMorphologyAssembler  
{  
  
    @Override  
    public MorphologySkeleton assemble(RegulatoryGraph graph) {  
        int n = graph.nodes().size();  
        int e = graph.edges().size();  
  
        Map<String, Double> p = new LinkedHashMap<String, Double>();  
        // toy parameters in [0,1] approx  
        p.put("nose_bridge_height", clamp01(0.2 + 0.1 * n));  
        p.put("jaw_width", clamp01(0.3 + 0.02 * e));  
        p.put("cheekbone_prominence", clamp01(0.25 + 0.05 * (n > 0 ? (double)  
e / (double) n : 0.0)));  
        return new MorphologySkeleton(p);  
    }  
  
    @Override  
    public FaceFamily project(MorphologySkeleton skeleton) {  
        Map<String, FaceFamily.Interval> env = new LinkedHashMap<String,  
FaceFamily.Interval>();  
        for (Map.Entry<String, Double> en : skeleton.parameters().entrySet())  
{  
            double v = en.getValue();  
            // toy: fixed uncertainty band  
            double band = 0.12;  
            env.put(en.getKey(), new FaceFamily.Interval(clamp01(v - band),  
clamp01(v + band)));  
        }  
        return new FaceFamily(env);  
    }  
  
    private static double clamp01(double x) {  
        return x < 0.0 ? 0.0 : (x > 1.0 ? 1.0 : x);  
    }  
}
```

com/dbm/bio/dna/morph/runtime/DefaultDna2MorphologyPipeline.java

```
package com.dbm.bio.dna.morph.runtime;

import com.dbm.bio.dna.morph.api.*;
import com.dbm.bio.dna.morph.ir.*;

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

public final class DefaultDna2MorphologyPipeline implements
IDna2MorphologyPipeline {

    private final ISnpProvider snpProvider;
    private final ISpanExtractor spanExtractor;
    private final ISnpBagBuilder snpBagBuilder;
    private final ICccDiscovery cccDiscovery;
    private final IRegulatoryGraphBuilder regulatoryGraphBuilder;
    private final IMorphologyAssembler morphologyAssembler;

    public DefaultDna2MorphologyPipeline(ISnpProvider snpProvider,
                                         ISpanExtractor spanExtractor,
                                         ISnpBagBuilder snpBagBuilder,
                                         ICccDiscovery cccDiscovery,
                                         IRegulatoryGraphBuilder
regulatoryGraphBuilder,
                                         IMorphologyAssembler
morphologyAssembler) {
        this.snpProvider = snpProvider;
        this.spanExtractor = spanExtractor;
        this.snpBagBuilder = snpBagBuilder;
        this.cccDiscovery = cccDiscovery;
        this.regulatoryGraphBuilder = regulatoryGraphBuilder;
        this.morphologyAssembler = morphologyAssembler;
    }

    @Override
    public Dna2MorphologyResult run(String sampleId, IDnaSource dna) {
        ExplainChain explain = new ExplainChain();

        List<DnaSpan> spans = spanExtractor.extractSpans(dna);
        explain.add(ev("SPAN", "Extracted spans=" + spans.size(), 0.5,
kv("dnaId", dna.id())));

        List<SnpCall> snps = snpProvider.loadSnps(sampleId);
        SnpBag snpBag = snpBagBuilder.build(sampleId, snps);
        explain.add(ev("SNP", "Loaded snps=" + snps.size(), 0.5,
kv("sampleId", sampleId)));

        List<CccCandidate> cccs = cccDiscovery.discover(spans, snpBag);
        explain.add(ev("CCC", "Discovered cccs=" + cccs.size(), 1.0,
kv("topCcc", cccs.isEmpty() ? "none" : cccs.get(0).cccKey())));

        RegulatoryGraph rg = regulatoryGraphBuilder.build(cccs);
```

```

        explain.add(ev("REG", "RegGraph nodes=" + rg.nodes().size() + ",
edges=" + rg.edges().size(), 0.8, null));

        MorphologySkeleton sk = morphologyAssembler.assemble(rg);
        explain.add(ev("ASM", "Skeleton params=" + sk.parameters().size(),
0.8, null));

        FaceFamily ff = morphologyAssembler.project(sk);
        explain.add(ev("PROJ", "FaceFamily envelopes=" +
ff.envelopes().size(), 0.8, null));

        // Meta: 越鱼控鱼越多 (方法论 evidence)
        explain.add(ev("META",
            "Structural control exposes more future-facing problems (越鱼
控鱼越多): CCC+rules+constraints reappear across domains.",
            0.2, null));

        return new Dna2MorphologyResult(sampleId, spans, snpBag, cccs, rg,
sk, ff, explain);
    }

    private static EvidenceItem ev(String kind, String msg, double cost,
Map<String, String> attrs) {
        return new EvidenceItem(kind, msg, cost, attrs);
    }

    private static Map<String, String> kv(String k, String v) {
        Map<String, String> m = new HashMap<String, String>();
        m.put(k, v);
        return m;
    }
}

```

5) Demo (可直接运行)

com/dbm/bio/dna/morph/demo/Dna2MorphologyDemo.java

```

package com.dbm.bio.dna.morph.demo;

import com.dbm.bio.dna.morph.api.IDna2MorphologyPipeline;
import com.dbm.bio.dna.morph.api.IDnaSource;
import com.dbm.bio.dna.morph.ir.Dna2MorphologyResult;
import com.dbm.bio.dna.morph.ir.EvidenceItem;
import com.dbm.bio.dna.morph.runtime.*;

public final class Dna2MorphologyDemo {

    public static void main(String[] args) {
        String sampleId = "SAMPLE_A";
        String dna = "ACGTACGTACGTATGACGTACGTACGTACGTATGACGTACGTACGTACGT";
    }
}

```

```

IDnaSource dnaSource = new StringDnaSource("dna:toy", dna);

IDna2MorphologyPipeline pipe = new DefaultDna2MorphologyPipeline(
    new MockSnpProvider(),
    new SimpleSpanExtractor(12),
    new SimpleSnpBagBuilder(),
    new SimpleCccDiscovery(),
    new SimpleRegulatoryGraphBuilder(),
    new SimpleMorphologyAssembler()
);

Dna2MorphologyResult r = pipe.run(sampleId, dnaSource);

System.out.println("=== DNA2Morphology Result ===");
System.out.println("sampleId=" + r.sampleId());
System.out.println("spans=" + r.spans().size());
System.out.println("snps=" + r.snpBag().snps().size());
System.out.println("cccs=" + r.cccs().size());
System.out.println("regGraph nodes=" +
r.regulatoryGraph().nodes().size() + " edges=" +
r.regulatoryGraph().edges().size());
System.out.println("skeleton=" + r.skeleton().parameters());
System.out.println("faceFamily=" + r.faceFamily().envelopes());
System.out.println("explain.totalCost=" +
r.explain().totalContributionCost());

    System.out.println("\n--- Explain Chain ---");
    for (EvidenceItem e : r.explain().items()) {
        System.out.println(e);
    }
}
}

```

6) JUnit4 玩具测试（验证关键不变量）

com/dbm/bio/dna/morph/test/Dna2MorphologyPipelineTest.java

```

package com.dbm.bio.dna.morph.test;

import com.dbm.bio.dna.morph.api.IDna2MorphologyPipeline;
import com.dbm.bio.dna.morph.api.IDnaSource;
import com.dbm.bio.dna.morph.ir.Dna2MorphologyResult;
import com.dbm.bio.dna.morph.runtime.*;
import org.junit.Assert;
import org.junit.Test;

public class Dna2MorphologyPipelineTest {

    private IDna2MorphologyPipeline newPipe() {
        return new DefaultDna2MorphologyPipeline(
            new MockSnpProvider(),
            new SimpleSpanExtractor(10),
            new SimpleSnpBagBuilder(),

```

```

        new SimpleCccDiscovery(),
        new SimpleRegulatoryGraphBuilder(),
        new SimpleMorphologyAssembler()
    );
}

@Test
public void testPipelineProducesNonEmptyIR() {
    IDnaSource dna = new StringDnaSource("dna:t1",
"ACGTATGACGTATGACGTACGTACGT");
    Dna2MorphologyResult r = newPipe().run("S1", dna);

    Assert.assertNotNull(r);
    Assert.assertTrue(r.spans().size() > 0);
    Assert.assertTrue(r.snpBag().snps().size() > 0);
    Assert.assertTrue(r.cccs().size() > 0);
    Assert.assertNotNull(r.regulatoryGraph());
    Assert.assertNotNull(r.skeleton());
    Assert.assertNotNull(r.faceFamily());
    Assert.assertNotNull(r.explain());
    Assert.assertTrue(r.explain().items().size() >= 5);
}

@Test
public void testFaceFamilyIntervalsAreValid() {
    IDnaSource dna = new StringDnaSource("dna:t2",
"ACGTACGTACGTACGTATGACGTACGT");
    Dna2MorphologyResult r = newPipe().run("S2", dna);

    for (com.dbm.bio.dna.morph.ir.FaceFamily.Interval in :
r.faceFamily().envelopes().values()) {
        Assert.assertTrue(in.hi() >= in.lo());
        Assert.assertTrue(in.lo() >= 0.0);
        Assert.assertTrue(in.hi() <= 1.0);
    }
}

@Test
public void testExplainChainCostIsPositive() {
    IDnaSource dna = new StringDnaSource("dna:t3",
"ATGATGATGATGATGACGTACGT");
    Dna2MorphologyResult r = newPipe().run("S3", dna);

    Assert.assertTrue(r.explain().totalContributionCost() > 0.0);
    boolean hasMeta = false;
    for (com.dbm.bio.dna.morph.ir.EvidenceItem e : r.explain().items()) {
        if ("META".equals(e.kind())) { hasMeta = true; break; }
    }
    Assert.assertTrue("Should include META evidence about 越鱼控鱼越多",
hasMeta);
}
}

```

7) 你后续最关键的“升级插槽”（我已预留接口位）

当你要从 MVP 升级到“真正 DBM DNA2Morphology”，优先替换三处即可：

1. SimpleCccDiscovery
 - 接入：GWAS priors / population stats / metric-tree CCC matching
2. SimpleRegulatoryGraphBuilder
 - 接入：non-coding 注释 + eQTL/TF binding + developmental stage model
3. SimpleMorphologyAssembler
 - 从参数字典升级为：landmarks topology graph + symmetry constraints + consistency solver

如果你希望我下一步继续“更 DBM 风味”地贴合你现有 repo（例如沿用你既有 Evidence/Ir/Runtime 命名、或接入你已有的 Metric Differential Tree / Span 基建），我可以在不改核心思路的前提下，给一版 **Drop-in 适配版**（重命名+包结构对齐+少量桥接类）。

兄弟，拓展的超级棒，谢谢了。进一步问题：脸型 CCC 是单元特征 CCC 的结构，有一个演化的问题：是在 IR 层（before CCCStarmap）处理 CCC 对单元特征 CCC 的结构（如 Event of Events），还是 CCCStarmap 之上构筑上层建筑算法？

你的问题可以直接落成一个工程决策结论：**两层都要做，但“结构定义的主战场”应在 IR 层完成；CCCStarmap 之上只做治理、压缩与跨样本对齐。**原因是：脸型 CCC 本质上是“可复用的结构单元”，如果不在 IR 层固化结构语义，后面所有 Starmap/距离/索引都会变成“把结构当噪声”。

下面我给出一个清晰的架构判断与落地方式。

结论：IR 层做“结构事实”，CCCStarmap 上做“结构治理”

A. IR 层（before CCCStarmap）应承担的职责：结构事实层

把“脸型 CCC = 单元特征 CCC 的结构”做成 IR 的一等公民，类似你说的 *Event of Events*。

在这一层做，等价于你在 Time-Series IR 里做过的：

- primitive events → composite events
- contribution 对齐
- 解释链可追溯

IR 层要输出：

1. 单元特征 CCC (primitive CCCs)
2. 结构组合 CCC (composite CCCs / CCCGraph)
3. 组合的可解释证据：边、门控、阶段、贡献成本

这样做的收益：

- 距离可计算（结构一致性可进入 metric）
- 索引可做（差分树能吃结构 token）
- 解释链天然可见（不会在 Starmap 上“补说明文”）
- 后续任何上层算法都能复用同一套“结构事实”

一句话：IR 层定义“什么是结构”。

B. CCCStarmap 之上应承担的职责：结构治理层

Starmap（或 CCCStarmap）更像是把 IR 变成可对齐、可比较、可检索的汇总投影，因此更适合做：

- 跨样本对齐与归一化（不同人群、不同测序平台）

- 结构压缩与签名化 (signature / hash / tier)
- 不确定性管理 (face family envelope、置信区间传播)
- 索引层加速 (top-k、ANN、树索引)
- 演化策略选择 (哪些 composite CCC 被“固化”为新 primitive CCC)

一句话：Starmap 层决定“如何使用结构”。

为什么“结构主战场在 IR 层”更 DBM、也更工程正确

1) 演化需要“可遗传单位”

演化的基本单位必须可复制、可变异、可比较。

如果你把组合结构放到 CCCStarmap 之上才构筑，那么：

- 组合结构会变成“搜索阶段的临时产物”
- 很难形成稳定的可复用构件
- 也就难以沉淀成可遗传的 CCC 单元

而放在 IR 层：

- composite CCC 是 IR 的实体
- 可以被记录、版本化、变异生成 family
- 可被下游多次使用与验证

2) 距离与解释链必须对齐

你们 DBM 的一贯硬标准：**distance** ↔ **evidence** ↔ **contributionCost** 对齐。

若结构只在 Starmap 上构建：

- 下游距离多半只能看“汇总向量/签名”

- 很难把“结构边、门控、阶段”精确回溯为贡献成本
- 解释链就会漂

放在 IR 层：

- 每条结构边（CCC→CCC gate）本身就是 evidence item
- contributionCost 可对齐到边与节点
- Starmap 只做投影，不改变因果账本

3) 索引层需要稳定 token

差分树、变尺寸块、签名索引都偏好“稳定 token”。

结构如果仅在 Starmap 上临时拼：

- token 变动大
- 索引不稳
- recall/precision 会发散

IR 层先固化 composite CCC：

- token 稳定
- 索引可积累
- 支持长期演化

推荐的分层实现（最小可落地）

IR 层新增两个核心 IR 实体

1. CccPrimitive（已有 CccCandidate 可以进化为此）
2. CccComposite（结构化 CCC，Event-of-Events）

建议结构：

- `CccComposite` 内含：
 - 子 CCC 列表 (node refs)
 - 边列表 (gate/weight/stage)
 - 局部一致性约束 (symmetry/topology hints)
 - 自己的 strength 与 contributionCost 分解

CCCStarmap 层新增两个对象

1. `CompositeSignature`：把 composite CCC 压缩成可比较签名 (分层 hash + tier)
2. `EvolutionLedger`：记录哪些 composite 被固化、哪些被淘汰、何时升级为 primitive

演化策略：何时把 composite “固化”为 primitive

这也是你问的“演化问题”的核心落点。建议一条可执行的规则：

当一个 `CccComposite` 满足：

- 跨样本复现率高 (support / coverage)
- 结构一致性高 (graph edit distance 小)
- 对输出形态贡献大且稳定 (contributionCost 稳)
- 可在不同人群/平台上对齐 (robust normalization)

则：

- 在 IR 层将其升级为 `CccPrimitiveVNext`
- 在 Starmap 层更新 signature 字典与索引

这对应 DBM 的“从组合结构沉淀出新概念”的机制。

你的问题的最终答案（可写进 ITEM 的那种）

- 结构本体应在 IR 层建模（Event-of-Events / CCCGraph / composite CCC），以保证可解释、可比较、可遗传、可索引。
- CCCStarmap 之上做治理与工程化（对齐、压缩、置信传播、索引加速、固化策略），以支持规模化检索与演化管理。

如果你愿意，我可以基于你前面那套 Java skeleton，直接补齐一组最小接口与类：

- CccComposite, CccEdge, CccGraphIR
- CompositeSignature, CompositeStarmapProjector
- EvolutionLedger, CompositePromotionPolicy

并加 2–3 个 JUnit4 玩具测试验证：

- composite 的 contributionCost 分解一致性
- signature 稳定性（small mutation 不破坏 tier）
- promotion 条件触发与版本升级链

讲的很好。同意你的观点。下一问：生物研究界的研究与知识，可以给这套算法提供高质量的切入点与标注，比全部冷启动要有效率的多，也能直接反馈给科研与应用？理想情况应该是 DBM 这套工具与生物研究进行有效的互动，以取得最佳的效果。

是的，而且这点在 DNA → Morphology CCC 这种问题上几乎是“成败分水岭”。原因很现实：该任务的信号极弱、组合效应强、且存在大量混杂因素；如果完全冷启动，工程上会在“相关性陷阱”和“不可复现”里消耗巨大成本。相反，生物研究界的既有知识可以把你们的 DBM 管线从一开始就拉到高信噪比区间，并且反过来把 DBM 的“结构化可解释输出”变成科研的增量工具。

下面我把“如何有效互动”拆成可执行的工程路径。

1) 生物知识能提供的高质量切入点

A. 先验约束与候选空间压缩（最值钱）

生物界的知识最强的作用不是“给答案”，而是把搜索空间压缩几个数量级，例如：

- 调控元件与非编码功能注释（enhancer/promoter/TF binding 等）
- eQTL / expression 相关信息（哪些变异影响哪些组织/时期的表达）
- 已知 GWAS hits（与面部形态、颅面发育相关位点/区域）
- 发育生物学的阶段性约束（哪类结构在何时形成、具有何种依赖）

对 DBM 来说，这些都可以直接进入 IR 层作为：

- SpanPrior（跨度先验）
- CccCandidatePrior（CCC 候选白名单/黑名单/权重）
- RegulatoryGateTemplates（门控模板）

B. 标注与弱标注（让 CCC “落地”）

形态学不一定需要像图像那样“逐像素标注”。更高性价比的是：

- 形态参数（landmarks、距离、角度、比率）
- 结构类别（下颌宽/窄、鼻梁高/低、颧骨突出/平等）
- 人群统计标签（族群、性别、年龄段）作为混杂控制项

这些可以作为 DBM 的监督信号：

- 用于 CCC discovery 的验证与筛选
- 用于 composite CCC 的稳定性评估
- 用于输出 FaceFamily 的置信区间校准

C. 可复现协议与负样本（工程里常被忽略，但科研很强）

科研界更擅长定义：

- 队列设计、分层、控制变量
- 负样本与对照组
- 多中心复现实验

这会直接提升 DBM 的“可持续迭代能力”，避免只在单一数据集上过拟合“局部真理”。

2) “DBM ↔ 生物研究”互动的理想闭环

把它表述成一个标准的双向闭环（你们 DBM 风格很适合）：

1. Bio Prior Ingestion（生物先验注入）

注释、已知位点、发育约束 → 转成 IR 可用的先验与 gate 模板

2. DBM CCC Discovery / Assembly（结构发现与组装）

输出：primitive CCC + composite CCC（结构边、门控、阶段、贡献成本）

3. Explainable Hypothesis Pack（可解释假设包）

对科研最有价值的是“可试验假设包”，包括：

- 候选调控路径（哪段非编码区通过哪条门控影响哪类形态）
- 关键节点/边的贡献排序（top-k evidence）
- 可复现统计（跨队列支持度）

4. Wet-lab / Bioinformatics Validation（验证）

验证结果回流：

- 支持：提升该 CCC/结构的 prior
- 否定：加入黑名单/惩罚项（防止反复踩坑）
- 部分支持：细分子群/阶段（更精细的 gate）

5. Promotion / Governance（固化与治理）

通过你前面认可的策略：把稳定的 composite CCC 升级为新的 primitive CCC（版本化），进入工具链长期资产。

这就是“DBM 工具与生物研究互相增益”的最短路径。

3) 工程落地建议：从 Day-1 就把接口留好

在你现有的 Java skeleton 上，我建议新增三类“科研互动插槽”，保证你不需要推翻现有管线：

A. IBioPriorProvider

- 输入：注释/先验（可来自文件、数据库、API）
- 输出：对 span/CCC/gate 的权重、白名单、黑名单、阶段模板

B. IPhenotypeLabelProvider

- 提供形态标签（连续参数 + 离散类）
- 支持弱标注与缺失值
- 同时输出混杂控制字段（sex/age/ancestry/batch）

C. IHypothesisExporter

- 把 DBM 结果导出成科研可用格式：
 - top-k CCC 与结构边
 - 可复现实证据（支持度/置信）
 - 建议验证实验类型（例如优先验证哪些 gate）

这样，DBM 不是“闭门造车的黑箱”，而是科研的“结构化假设生成器”。

4) 这会如何“直接反馈科研与应用”

对科研的直接贡献：

- 把海量遗传变异从“统计相关”推进到“结构路径假设”
- 将非编码区从“难解释噪声”提升为“规则层候选”
- 强制输出可检验假设，天然鼓励复现与反证

对应用的直接贡献：

- 输出人脸“族群/形态族”的置信包络，而非虚假确定性
- 能解释“为什么推断如此”，便于司法/寻亲等场景的合规与审查
- 随着验证回流，系统的可信度与覆盖率会稳定提升

5) 最关键的一点：你们 DBM 的优势正好补科研短板

科研常见短板是：

- 碎片化：点位统计多，结构路径少
- 难整合：多组学、多队列、多阶段难统一表达
- 难解释：机制链条断裂

DBM 强项正是：

- CCC + 结构组装
- 规则/门控层表达
- 解释链与贡献成本对齐
- “概念固化与版本化”的演化机制

所以这不是“借用生物标注来做工程”，而是**互补式协同**：生物给先验与验证，DBM 给结构化发现与假设生成。

如果你愿意，我可以下一步直接把这套“科研互动闭环”固化成代码插槽（Java 8）并补一个最小 Demo：

- `FileBioPriorProvider`（从 JSON/YAML 读先验）
- `ToyPhenotypeLabelProvider`（给 3 个形态标签）
- `HypothesisExporter`（输出 top-k CCC + edges + evidence 到控制台/JSON）
- 2 个 JUnit4：验证“prior 能改变 CCC 排名 / 输出假设包稳定可复现”

你一句话我就按你 repo 风格继续补齐。

讲的很好。请直接把这套“科研互动闭环”固化成代码插槽（Java 8）并补一个最小

Demo： `FileBioPriorProvider`（从 JSON/YAML 读先验） `ToyPhenotypeLabelProvider`（给 3 个形态标签） `HypothesisExporter`（输出 top-k CCC + edges + evidence 到控制台/JSON） 2 个 JUnit4：验证“prior 能改变 CCC 排名 / 输出假设包稳定可复现”

Thought for 18s

下面是你要的“科研互动闭环”代码插槽（Java 8）+ 最小 Demo + 2 个 JUnit4，在你前面 DNA2Morphology skeleton 基础上增量扩展，不推翻既有设计。

设计原则（符合 DBM 风味）：

- **BioPrior** 是 *IR 层之前/之中的*“候选空间压缩 + 权重注入”
- **Phenotype Labels** 是 *验证与校准*的外部监督信号（弱标注友好）
- **Hypothesis Exporter** 输出“科研可用假设包”：top-k CCC、regulatory edges、evidence chain（可控 JSON）

解析：实现一个 MVP 级 JSON/YAML 子集解析器（无第三方依赖）。支持你们配置所需的最小结构（Map / List / 数值 / 字符串）。后续你可无缝替换为 Jackson/SnakeYAML。

A) 新增 API：科研互动插槽

com/dbm/bio/dna/morph/api/IBioPriorProvider.java

```
package com.dbm.bio.dna.morph.api;

import com.dbm.bio.dna.morph.ir.BioPriorBundle;

/**
 * Loads domain priors from bio research knowledge (GWAS hits, regulatory
 * annotations, etc).
 * MVP: file-based JSON/YAML; production: DB/API backed.
 */
public interface IBioPriorProvider {
    BioPriorBundle load(String sampleId);
}
```

com/dbm/bio/dna/morph/api/IPhenotypeLabelProvider.java

```
package com.dbm.bio.dna.morph.api;

import com.dbm.bio.dna.morph.ir.PhenotypeLabels;

/**
 * Provides phenotype labels (continuous or categorical) + confounders.
 * MVP: toy provider; production: cohort metadata / landmark pipeline.
 */
public interface IPhenotypeLabelProvider {
    PhenotypeLabels load(String sampleId);
}
```

com/dbm/bio/dna/morph/api/IHypothesisExporter.java

```
package com.dbm.bio.dna.morph.api;

import com.dbm.bio.dna.morph.ir.Dna2MorphologyResult;

public interface IHypothesisExporter {

    /**
     * Export top-k CCC + edges + evidence into a JSON string (and/or
     * console).
     * Must be deterministic for reproducibility.
     */
    String export(Dna2MorphologyResult result, int topK, boolean
        printToConsole);
}
```

B) 新增 IR : BioPrior 与 PhenotypeLabels

com/dbm/bio/dna/morph/ir/BioPriorBundle.java

```
package com.dbm.bio.dna.morph.ir;
```

```

import java.util.Collections;
import java.util.LinkedHashMap;
import java.util.Map;

/**
 * Minimal priors used by the pipeline:
 * - cccWeights: multiplicative or additive boosts per CCC key
 * - edgeWeights: boost/penalty for regulatory edges "from->to"
 */
public final class BioPriorBundle {

    private final Map<String, Double> cccWeights;
    private final Map<String, Double> edgeWeights;
    private final String sourceTag;

    public BioPriorBundle(Map<String, Double> cccWeights, Map<String, Double>
edgeWeights, String sourceTag) {
        this.cccWeights = (cccWeights == null) ? Collections.<String,
Double>emptyMap()
            : Collections.unmodifiableMap(new LinkedHashMap<String,
Double>(cccWeights));
        this.edgeWeights = (edgeWeights == null) ? Collections.<String,
Double>emptyMap()
            : Collections.unmodifiableMap(new LinkedHashMap<String,
Double>(edgeWeights));
        this.sourceTag = sourceTag;
    }

    public Map<String, Double> cccWeights() { return cccWeights; }
    public Map<String, Double> edgeWeights() { return edgeWeights; }
    public String sourceTag() { return sourceTag; }

    public double cccWeightOrDefault(String cccKey, double def) {
        Double v = cccWeights.get(cccKey);
        return v == null ? def : v.doubleValue();
    }

    public double edgeWeightOrDefault(String from, String to, double def) {
        String k = from + "->" + to;
        Double v = edgeWeights.get(k);
        return v == null ? def : v.doubleValue();
    }
}

```

com/dbm/bio/dna/morph/ir/PhenotypeLabels.java

```

package com.dbm.bio.dna.morph.ir;

import java.util.Collections;
import java.util.LinkedHashMap;
import java.util.Map;

/**
 * MVP phenotype labels + confounders.
 * Values are double for simplicity (categorical can be encoded).
 */

```

```

public final class PhenotypeLabels {

    private final Map<String, Double> labels;
    private final Map<String, String> confounders; // sex/age/ancestry/batch
    etc.

    public PhenotypeLabels(Map<String, Double> labels, Map<String, String>
    confounders) {
        this.labels = (labels == null) ? Collections.<String,
    Double>emptyMap()
        : Collections.unmodifiableMap(new LinkedHashMap<String,
    Double>(labels));
        this.confounders = (confounders == null) ? Collections.<String,
    String>emptyMap()
        : Collections.unmodifiableMap(new LinkedHashMap<String,
    String>(confounders));
    }

    public Map<String, Double> labels() { return labels; }
    public Map<String, String> confounders() { return confounders; }
}

```

C) FileBioPriorProvider : JSON/YAML 读先验（无外部依赖）

支持的最小配置结构（JSON 或 YAML 都可）：

JSON 示例

```

{
  "sourceTag": "toy-prior-v1",
  "cccWeights": {
    "CCC:JAW_WIDTH:V1": 3.0,
    "CCC:NOSE_BRIDGE:V1": 1.0
  },
  "edgeWeights": {
    "CCC:JAW_WIDTH:V1->CCC:CHEEKBONE:V1": 2.0
  }
}

```

YAML 示例

```

sourceTag: toy-prior-v1
cccWeights:
  CCC:JAW_WIDTH:V1: 3.0
  CCC:NOSE_BRIDGE:V1: 1.0
edgeWeights:
  "CCC:JAW_WIDTH:V1->CCC:CHEEKBONE:V1": 2.0

```

com/dbm/bio/dna/morph/runtime/FileBioPriorProvider.java

```
package com.dbm.bio.dna.morph.runtime;

import com.dbm.bio.dna.morph.api.IBioPriorProvider;
import com.dbm.bio.dna.morph.ir.BioPriorBundle;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.LinkedHashMap;
import java.util.Map;

/**
 * MVP JSON/YAML parser (subset) without third-party libs.
 * Supports the exact structure needed for priors.
 */
public final class FileBioPriorProvider implements IBioPriorProvider {

    private final File file;

    public FileBioPriorProvider(File file) {
        this.file = file;
    }

    @Override
    public BioPriorBundle load(String sampleId) {
        String text = readAll(file);
        text = text.trim();
        if (text.startsWith("{")) {
            return parseJson(text);
        }
        return parseYaml(text);
    }

    private static String readAll(File f) {
        try {
            BufferedReader br = new BufferedReader(new FileReader(f));
            StringBuilder sb = new StringBuilder();
            String line;
            while ((line = br.readLine()) != null) {
                sb.append(line).append("\n");
            }
            br.close();
            return sb.toString();
        } catch (Exception e) {
            throw new RuntimeException("Failed to read prior file: " + f, e);
        }
    }

    // ---- JSON (very small subset parser) ----
    private static BioPriorBundle parseJson(String json) {
        // Extremely small parser:
        // - extract "sourceTag":"..."
        // - extract object blocks for cccWeights and edgeWeights with
        string->number entries
    }
}
```

```

        String sourceTag = extractJsonString(json, "sourceTag");
        Map<String, Double> ccc = extractJsonMapStringDouble(json,
"cccWeights");
        Map<String, Double> edge = extractJsonMapStringDouble(json,
"edgeWeights");
        return new BioPriorBundle(ccc, edge, sourceTag);
    }

    private static String extractJsonString(String json, String key) {
        String k = "\"" + key + "\"";
        int p = json.indexOf(k);
        if (p < 0) return null;
        int colon = json.indexOf(':', p);
        if (colon < 0) return null;
        int q1 = json.indexOf('"', colon + 1);
        if (q1 < 0) return null;
        int q2 = json.indexOf('"', q1 + 1);
        if (q2 < 0) return null;
        return json.substring(q1 + 1, q2);
    }

    private static Map<String, Double> extractJsonMapStringDouble(String
json, String key) {
        Map<String, Double> out = new LinkedHashMap<String, Double>();
        String k = "\"" + key + "\"";
        int p = json.indexOf(k);
        if (p < 0) return out;

        int brace1 = json.indexOf('{', p);
        if (brace1 < 0) return out;
        int brace2 = findMatchingBrace(json, brace1);
        if (brace2 < 0) return out;

        String body = json.substring(brace1 + 1, brace2).trim();
        // entries like: "AAA": 1.2, "BBB": 3
        // split by commas not inside quotes (MVP: assume no commas in keys)
        String[] parts = body.split(",");
        for (String part : parts) {
            String s = part.trim();
            if (s.isEmpty()) continue;
            int q1 = s.indexOf('"');
            int q2 = s.indexOf('"', q1 + 1);
            if (q1 < 0 || q2 < 0) continue;
            String mapKey = s.substring(q1 + 1, q2);
            int colon = s.indexOf(':', q2);
            if (colon < 0) continue;
            String num = s.substring(colon + 1).trim();
            // strip possible quotes (rare)
            num = num.replace("\"", "");
            try {
                out.put(mapKey, Double.valueOf(num));
            } catch (Exception ignore) {
                // ignore invalid numbers in MVP
            }
        }
        return out;
    }
}

```



```

private static int findMatchingBrace(String s, int openPos) {
    int depth = 0;
    for (int i = openPos; i < s.length(); i++) {
        char c = s.charAt(i);
        if (c == '{') depth++;
        else if (c == '}') {
            depth--;
            if (depth == 0) return i;
        }
    }
    return -1;
}

// ---- YAML (very small subset parser) ----
private static BioPriorBundle parseYaml(String yaml) {
    String sourceTag = null;
    Map<String, Double> ccc = new LinkedHashMap<String, Double>();
    Map<String, Double> edge = new LinkedHashMap<String, Double>();

    String section = null; // "cccWeights" | "edgeWeights" | null

    String[] lines = yaml.split("\n");
    for (String raw : lines) {
        String line = raw.trim();
        if (line.isEmpty() || line.startsWith("#")) continue;

        if (line.startsWith("sourceTag:")) {
            sourceTag = line.substring("sourceTag:".length()).trim();
            sourceTag = stripQuotes(sourceTag);
            continue;
        }
        if (line.startsWith("cccWeights:")) { section = "cccWeights";
continue; }
        if (line.startsWith("edgeWeights:")) { section = "edgeWeights";
continue; }

        // key: value entries under section
        int idx = line.indexOf(':');
        if (idx < 0) continue;
        String k = stripQuotes(line.substring(0, idx).trim());
        String v = stripQuotes(line.substring(idx + 1).trim());
        if (v.isEmpty()) continue;

        try {
            double d = Double.parseDouble(v);
            if ("cccWeights".equals(section)) ccc.put(k, d);
            else if ("edgeWeights".equals(section)) edge.put(k, d);
        } catch (Exception ignore) {}
    }

    return new BioPriorBundle(ccc, edge, sourceTag);
}

private static String stripQuotes(String s) {
    if (s == null) return null;

```

```

        s = s.trim();
        if (s.startsWith("\"") && s.endsWith("\"") && s.length() >= 2) {
            return s.substring(1, s.length() - 1);
        }
        return s;
    }
}

```

D) ToyPhenotypeLabelProvider : 3 个形态标签 (MVP)

com/dbm/bio/dna/morph/runtime/ToyPhenotypeLabelProvider.java

```

package com.dbm.bio.dna.morph.runtime;

import com.dbm.bio.dna.morph.api.IPhenotypeLabelProvider;
import com.dbm.bio.dna.morph.ir.PhenotypeLabels;

import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Random;

/**
 * MVP: deterministic toy labels per sampleId.
 * Labels in [0,1]. Confounders included for future gating/calibration.
 */
public final class ToyPhenotypeLabelProvider implements
IPhenotypeLabelProvider {

    @Override
    public PhenotypeLabels load(String sampleId) {
        Random r = new Random(sampleId == null ? 0 : sampleId.hashCode());

        Map<String, Double> labels = new LinkedHashMap<String, Double>();
        labels.put("nose_bridge_height", clamp01(0.3 + 0.4 *
r.nextDouble()));
        labels.put("jaw_width", clamp01(0.2 + 0.5 * r.nextDouble()));
        labels.put("cheekbone_prominence", clamp01(0.25 + 0.5 *
r.nextDouble()));

        Map<String, String> conf = new LinkedHashMap<String, String>();
        conf.put("sex", (r.nextBoolean() ? "M" : "F"));
        conf.put("ancestry", (r.nextInt(3) == 0 ? "EAS" : (r.nextInt(2) ==
0 ? "EUR" : "AFR")));
        conf.put("batch", "toy");

        return new PhenotypeLabels(labels, conf);
    }

    private static double clamp01(double x) {
        return x < 0.0 ? 0.0 : (x > 1.0 ? 1.0 : x);
    }
}

```

E) HypothesisExporter : top-k CCC + edges + evidence → console/JSON (稳定可复现)

com/dbm/bio/dna/morph/runtime/HypothesisExporter.java

```
package com.dbm.bio.dna.morph.runtime;

import com.dbm.bio.dna.morph.api.IHypothesisExporter;
import com.dbm.bio.dna.morph.ir.*;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.Map;

/**
 * Deterministic exporter for research consumption:
 * - top-k CCCs by strength (desc), tie-break by key (asc)
 * - regulatory edges sorted lexicographically
 * - evidence items in original order (pipeline order)
 */
public final class HypothesisExporter implements IHypothesisExporter {

    @Override
    public String export(Dna2MorphologyResult result, int topK, boolean
        printToConsole) {
        String json = toJson(result, topK);
        if (printToConsole) {
            System.out.println(json);
        }
        return json;
    }

    private String toJson(Dna2MorphologyResult r, int topK) {
        StringBuilder sb = new StringBuilder();
        sb.append("{");

        kv(sb, "sampleId", r.sampleId()); sb.append(",");
        kv(sb, "topK", String.valueOf(topK)); sb.append(",");

        // top CCCs
        List<CccCandidate> cccs = new ArrayList<CccCandidate>(r.cccs());
        Collections.sort(cccs, new Comparator<CccCandidate>() {
            @Override public int compare(CccCandidate a, CccCandidate b) {
                int c = Double.compare(b.strength(), a.strength());
                if (c != 0) return c;
                return a.cccKey().compareTo(b.cccKey());
            }
        });

        sb.append("\ntopCccs\":[");
```

```

int k = Math.min(topK, cccs.size());
for (int i = 0; i < k; i++) {
    CccCandidate c = cccs.get(i);
    if (i > 0) sb.append(",");
    sb.append("{");
    kv(sb, "cccKey", c.cccKey()); sb.append(",");
    kv(sb, "strength", formatDouble(c.strength()));
    sb.append("}");
}
sb.append("],");

// edges
List<RegulatoryGraph.Edge> edges = new
ArrayList<RegulatoryGraph.Edge>(r.regulatoryGraph().edges());
Collections.sort(edges, new Comparator<RegulatoryGraph.Edge>() {
    @Override public int compare(RegulatoryGraph.Edge a,
RegulatoryGraph.Edge b) {
        String ka = a.from() + "->" + a.to() + "@" + a.gate();
        String kb = b.from() + "->" + b.to() + "@" + b.gate();
        return ka.compareTo(kb);
    }
});

sb.append("\nedges\":[");
for (int i = 0; i < edges.size(); i++) {
    RegulatoryGraph.Edge e = edges.get(i);
    if (i > 0) sb.append(",");
    sb.append("{");
    kv(sb, "from", e.from()); sb.append(",");
    kv(sb, "to", e.to()); sb.append(",");
    kv(sb, "weight", formatDouble(e.weight())); sb.append(",");
    kv(sb, "gate", e.gate());
    sb.append("}");
}
sb.append("],");

// evidence
sb.append("\nevidence\":[");
List<EvidenceItem> evs = r.explain().items();
for (int i = 0; i < evs.size(); i++) {
    EvidenceItem e = evs.get(i);
    if (i > 0) sb.append(",");
    sb.append("{");
    kv(sb, "kind", e.kind()); sb.append(",");
    kv(sb, "contributionCost", formatDouble(e.contributionCost()));
sb.append(",");
    kv(sb, "message", e.message()); sb.append(",");
    sb.append("\nattrs\":[");
    boolean first = true;
    for (Map.Entry<String, String> en : e.attrs().entrySet()) {
        if (!first) sb.append(",");
        first = false;
        kv(sb, en.getKey(), en.getValue());
    }
    sb.append("}");
    sb.append("}");
}
}

```

```

        sb.append("]");

        sb.append("}");
        return sb.toString();
    }

    private static void kv(StringBuilder sb, String k, String v) {
        sb.append("\"").append(escape(k)).append("\":");
        if (isNumberLiteral(v)) {
            sb.append(v);
        } else {
            sb.append("\"").append(escape(v)).append("\"");
        }
    }

    private static boolean isNumberLiteral(String s) {
        if (s == null || s.isEmpty()) return false;
        char c0 = s.charAt(0);
        return (c0 == '-' || (c0 >= '0' && c0 <= '9'));
    }

    private static String formatDouble(double d) {
        // stable formatting
        return String.format(java.util.Locale.US, "%.6f", d);
    }

    private static String escape(String s) {
        if (s == null) return "";
        return s.replace("\\", "\\").replace("\'", "\\\'").replace("\n",
            "\\n").replace("\r", "\\r");
    }
}

```

F) 将 Prior 注入 CCC 排名与 Edge 权重：可插拔增强版 CCC/RegBuilder

1) 新接口：带 prior 的 CCC discovery（保持兼容）

新增一个 wrapper，不改原 `ICccDiscovery` 接口；在 pipeline 内部组合即可。

com/dbm/bio/dna/morph/runtime/PriorAwareCccDiscovery.java

```

package com.dbm.bio.dna.morph.runtime;

import com.dbm.bio.dna.morph.api.ICccDiscovery;
import com.dbm.bio.dna.morph.ir.*;

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

```

```

/**
 * Wraps an existing ICccDiscovery and applies BioPriorBundle
 * by adjusting CCC strength deterministically.
 */
public final class PriorAwareCccDiscovery implements ICccDiscovery {

    private final ICccDiscovery base;
    private final BioPriorBundle prior;

    public PriorAwareCccDiscovery(ICccDiscovery base, BioPriorBundle prior) {
        this.base = base;
        this.prior = prior;
    }

    @Override
    public List<CccCandidate> discover(List<DnaSpan> spans, SnpBag snpBag) {
        List<CccCandidate> raw = base.discover(spans, snpBag);
        if (prior == null || prior.cccWeights().isEmpty()) return raw;

        List<CccCandidate> out = new ArrayList<CccCandidate>(raw.size());
        for (CccCandidate c : raw) {
            double w = prior.cccWeightOrDefault(c.cccKey(), 1.0); //
multiplicative
            double newStrength = c.strength() * w;

            out.add(new CccCandidate(
                c.cccKey(),
                newStrength,
                c.supportingSpans(),
                c.supportingSnps()
            ));
        }
        return out;
    }
}

```

2) Prior 注入 edge 权重：RegGraph builder wrapper

com/dbm/bio/dna/morph/runtime/PriorAwareRegulatoryGraphBuilder.java

```

package com.dbm.bio.dna.morph.runtime;

import com.dbm.bio.dna.morph.api.IRegulatoryGraphBuilder;
import com.dbm.bio.dna.morph.ir.*;

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

/**
 * Applies BioPriorBundle edge weights as multiplicative boosts.
 */
public final class PriorAwareRegulatoryGraphBuilder implements
IRegulatoryGraphBuilder {

```

```

    private final IRegulatoryGraphBuilder base;
    private final BioPriorBundle prior;

    public PriorAwareRegulatoryGraphBuilder(IRegulatoryGraphBuilder base,
    BioPriorBundle prior) {
        this.base = base;
        this.prior = prior;
    }

    @Override
    public RegulatoryGraph build(List<CccCandidate> cccs) {
        RegulatoryGraph g = base.build(cccs);
        if (prior == null || prior.edgeWeights().isEmpty()) return g;

        // Rebuild edges applying weight boosts (RegulatoryGraph in MVP
stores edges in list only)
        RegulatoryGraph out = new RegulatoryGraph();
        for (String n : g.nodes()) out.addNode(n);

        List<RegulatoryGraph.Edge> edges = new
ArrayList<RegulatoryGraph.Edge>(g.edges());
        for (RegulatoryGraph.Edge e : edges) {
            double w = prior.edgeWeightOrDefault(e.from(), e.to(), 1.0);
            out.addEdge(new RegulatoryGraph.Edge(e.from(), e.to(), e.weight()
* w, e.gate()));
        }
        return out;
    }
}

```

G) 闭环 Pipeline：把 Prior、Label、Exporter 都接入（不破坏原 pipeline）

com/dbm/bio/dna/morph/runtime/ResearchLoopPipeline.java

```

package com.dbm.bio.dna.morph.runtime;

import com.dbm.bio.dna.morph.api.*;
import com.dbm.bio.dna.morph.ir.*;

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

/**
 * Research interaction loop:
 * - load priors + phenotype labels
 * - run pipeline with prior-aware CCC + prior-aware regulatory graph
 * - export hypothesis pack (top-k CCC + edges + evidence)
 */
public final class ResearchLoopPipeline implements IDna2MorphologyPipeline {

```

```

private final IBioPriorProvider priorProvider;
private final IPhenotypeLabelProvider labelProvider;
private final IHypothesisExporter exporter;

private final ISnpProvider snpProvider;
private final ISpanExtractor spanExtractor;
private final ISnpBagBuilder snpBagBuilder;
private final ICccDiscovery baseCccDiscovery;
private final IRegulatoryGraphBuilder baseRegGraphBuilder;
private final IMorphologyAssembler morphologyAssembler;

private final int exportTopK;
private final boolean printExport;

public ResearchLoopPipeline(IBioPriorProvider priorProvider,
                           IPhenotypeLabelProvider labelProvider,
                           IHypothesisExporter exporter,
                           ISnpProvider snpProvider,
                           ISpanExtractor spanExtractor,
                           ISnpBagBuilder snpBagBuilder,
                           ICccDiscovery baseCccDiscovery,
                           IRegulatoryGraphBuilder baseRegGraphBuilder,
                           IMorphologyAssembler morphologyAssembler,
                           int exportTopK,
                           boolean printExport) {
    this.priorProvider = priorProvider;
    this.labelProvider = labelProvider;
    this.exporter = exporter;
    this.snpProvider = snpProvider;
    this.spanExtractor = spanExtractor;
    this.snpBagBuilder = snpBagBuilder;
    this.baseCccDiscovery = baseCccDiscovery;
    this.baseRegGraphBuilder = baseRegGraphBuilder;
    this.morphologyAssembler = morphologyAssembler;
    this.exportTopK = exportTopK;
    this.printExport = printExport;
}

@Override
public Dna2MorphologyResult run(String sampleId, IDnaSource dna) {

    BioPriorBundle prior = (priorProvider == null) ? null :
priorProvider.load(sampleId);
    PhenotypeLabels labels = (labelProvider == null) ? null :
labelProvider.load(sampleId);

    ExplainChain explain = new ExplainChain();
    if (prior != null) {
        explain.add(new EvidenceItem("PRIOR",
            "Loaded BioPriorBundle sourceTag=" + prior.sourceTag()
            + " cccWeights=" + prior.cccWeights().size()
            + " edgeWeights=" + prior.edgeWeights().size(),
            0.2, kv("sourceTag", prior.sourceTag())));
    }
    if (labels != null) {
        explain.add(new EvidenceItem("LABEL",

```



```

        "Loaded PhenotypeLabels labels=" + labels.labels().size()
        + " confounders=" + labels.confounders().size(),
        0.2, kv("sex", labels.confounders().get("sex"))));
    }

    List<DnaSpan> spans = spanExtractor.extractSpans(dna);
    explain.add(new EvidenceItem("SPAN", "Extracted spans=" +
    spans.size(), 0.5, kv("dnaId", dna.id())));

    List<SnpCall> snps = snpProvider.loadSnps(sampleId);
    SnpBag snpBag = snpBagBuilder.build(sampleId, snps);
    explain.add(new EvidenceItem("SNP", "Loaded snps=" + snps.size(),
    0.5, kv("sampleId", sampleId)));

    ICccDiscovery cccDiscovery = (prior == null) ? baseCccDiscovery : new
    PriorAwareCccDiscovery(baseCccDiscovery, prior);
    List<CccCandidate> cccs = cccDiscovery.discover(spans, snpBag);
    explain.add(new EvidenceItem("CCC", "Discovered cccs=" + cccs.size(),
    1.0, kv("priorApplied", String.valueOf(prior != null))));

    IRegulatoryGraphBuilder regBuilder = (prior == null) ?
    baseRegGraphBuilder : new
    PriorAwareRegulatoryGraphBuilder(baseRegGraphBuilder, prior);
    RegulatoryGraph rg = regBuilder.build(cccs);
    explain.add(new EvidenceItem("REG", "RegGraph nodes=" +
    rg.nodes().size() + ", edges=" + rg.edges().size(), 0.8, null));

    MorphologySkeleton sk = morphologyAssembler.assemble(rg);
    explain.add(new EvidenceItem("ASM", "Skeleton params=" +
    sk.parameters().size(), 0.8, null));

    FaceFamily ff = morphologyAssembler.project(sk);
    explain.add(new EvidenceItem("PROJ", "FaceFamily envelopes=" +
    ff.envelopes().size(), 0.8, null));

    // Methodology meta evidence
    explain.add(new EvidenceItem("META",
        "Structural control exposes more future-facing problems (越鱼
    控鱼越多): CCC+rules+constraints reappear across domains.",
        0.2, null));

    Dna2MorphologyResult result = new Dna2MorphologyResult(sampleId,
    spans, snpBag, cccs, rg, sk, ff, explain);

    if (exporter != null) {
        String json = exporter.export(result, exportTopK, printExport);
        explain.add(new EvidenceItem("EXPORT",
            "Exported hypothesisPack bytes=" + (json == null ? 0 :
    json.length()),
            0.1, null));
    }

    return result;
}

private static Map<String, String> kv(String k, String v) {

```

```

        Map<String, String> m = new HashMap<String, String>();
        if (k != null && v != null) m.put(k, v);
        return m;
    }
}

```

H) 最小 Demo：从 JSON/YAML prior 读入，输出 hypothesis JSON

com/dbm/bio/dna/morph/demo/ResearchLoopDemo.java

```

package com.dbm.bio.dna.morph.demo;

import com.dbm.bio.dna.morph.api.*;
import com.dbm.bio.dna.morph.ir.Dna2MorphologyResult;
import com.dbm.bio.dna.morph.runtime.*;

import java.io.File;
import java.io.FileWriter;

public final class ResearchLoopDemo {

    public static void main(String[] args) throws Exception {
        String sampleId = "SAMPLE_A";
        String dna = "ACGTACGTACGTATGACGTACGTACGTACGTATGACGTACGTACGTACGT";

        // write a small JSON prior file for demo
        File priorFile = File.createTempFile("dbm_prior_", ".json");
        FileWriter fw = new FileWriter(priorFile);
        fw.write("{\n");
        fw.write("  \"sourceTag\": \"toy-prior-v1\", \n");
        fw.write("  \"cccWeights\": {\n");
        fw.write("    \"CCC:JAW_WIDTH:V1\": 3.0, \n");
        fw.write("    \"CCC:NOSE_BRIDGE:V1\": 1.0 \n");
        fw.write("  }, \n");
        fw.write("  \"edgeWeights\": {\n");
        fw.write("    \"CCC:JAW_WIDTH:V1->CCC:CHEEKBONE:V1\": 2.0 \n");
        fw.write("  } \n");
        fw.write("}\n");
        fw.close();

        IBioPriorProvider priorProvider = new
FileBioPriorProvider(priorFile);
        IPhenotypeLabelProvider labelProvider = new
ToyPhenotypeLabelProvider();
        IHypothesisExporter exporter = new HypothesisExporter();

        IDnaSource dnaSource = new StringDnaSource("dna:toy", dna);

        IDna2MorphologyPipeline pipe = new ResearchLoopPipeline(
            priorProvider,
            labelProvider,

```

```

        exporter,
        new MockSnpProvider(),
        new SimpleSpanExtractor(12),
        new SimpleSnpBagBuilder(),
        new SimpleCccDiscovery(),
        new SimpleRegulatoryGraphBuilder(),
        new SimpleMorphologyAssembler(),
        2,
        true
    );

    Dna2MorphologyResult r = pipe.run(sampleId, dnaSource);
    System.out.println("Done. CCC count=" + r.cccs().size() + " edges=" +
r.regulatoryGraph().edges().size());
    System.out.println("Prior file: " + priorFile.getAbsolutePath());
}
}

```

I) JUnit4：两条断言

1) prior 能改变 CCC 排名

com/dbm/bio/dna/morph/test/PriorAffectsCccRankingTest.java

```

package com.dbm.bio.dna.morph.test;

import com.dbm.bio.dna.morph.api.*;
import com.dbm.bio.dna.morph.ir.CccCandidate;
import com.dbm.bio.dna.morph.ir.Dna2MorphologyResult;
import com.dbm.bio.dna.morph.runtime.*;
import org.junit.Assert;
import org.junit.Test;

import java.io.File;
import java.io.FileWriter;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class PriorAffectsCccRankingTest {

    @Test
    public void testPriorChangesTopCcc() throws Exception {
        String sampleId = "S_RANK";
        IDnaSource dna = new StringDnaSource("dna:t_rank",
"ACGTATGACGTATGACGTACGTACGT");

        // Prior boosts JAW_WIDTH strongly
        File priorFile = File.createTempFile("dbm_prior_rank_", ".yaml");
        FileWriter fw = new FileWriter(priorFile);
    }
}

```

```

        fw.write("sourceTag: rank-test\n");
        fw.write("cccWeights:\n");
        fw.write("  CCC:JAW_WIDTH:V1: 10.0\n");
        fw.write("  CCC:NOSE_BRIDGE:V1: 1.0\n");
        fw.close();

        IBioPriorProvider priorProvider = new
FileBioPriorProvider(priorFile);
        IPhenotypeLabelProvider labelProvider = new
ToyPhenotypeLabelProvider();

        IDna2MorphologyPipeline pipe = new ResearchLoopPipeline(
            priorProvider,
            labelProvider,
            null, // no exporter
            new MockSnpProvider(),
            new SimpleSpanExtractor(10),
            new SimpleSnpBagBuilder(),
            new SimpleCccDiscovery(),
            new SimpleRegulatoryGraphBuilder(),
            new SimpleMorphologyAssembler(),
            2,
            false
        );

        Dna2MorphologyResult r = pipe.run(sampleId, dna);

        // Determine top CCC by strength
        List<CccCandidate> cccs = new ArrayList<CccCandidate>(r.cccs());
        Collections.sort(cccs, new Comparator<CccCandidate>() {
            @Override public int compare(CccCandidate a, CccCandidate b) {
                int c = Double.compare(b.strength(), a.strength());
                if (c != 0) return c;
                return a.cccKey().compareTo(b.cccKey());
            }
        });

        Assert.assertFalse(cccs.isEmpty());
        Assert.assertEquals("CCC:JAW_WIDTH:V1", cccs.get(0).cccKey());
    }
}

```

2) 输出假设包稳定可复现（同输入同 prior → JSON 必须一致）

com/dbm/bio/dna/morph/test/HypothesisPackDeterminismTest.java

```

package com.dbm.bio.dna.morph.test;

import com.dbm.bio.dna.morph.api.*;
import com.dbm.bio.dna.morph.ir.Dna2MorphologyResult;
import com.dbm.bio.dna.morph.runtime.*;
import org.junit.Assert;
import org.junit.Test;

import java.io.File;

```

```

import java.io.FileWriter;

public class HypothesisPackDeterminismTest {

    @Test
    public void testExporterIsDeterministic() throws Exception {
        String sampleId = "S_DET";
        IDnaSource dna = new StringDnaSource("dna:t_det",
"ATGATGATGACGTACGTACGTATGACGT");

        File priorFile = File.createTempFile("dbm_prior_det_", ".json");
        FileWriter fw = new FileWriter(priorFile);
        fw.write("{\n");
        fw.write("  \"sourceTag\": \"det-test\", \n");
        fw.write("  \"cccWeights\": {\"CCC:CHEEKBONE:V1\": 2.0}, \n");
        fw.write("  \"edgeWeights\": {\"CCC:CHEEKBONE:V1->CCC:JAW_WIDTH:V1\":
1.5}\n");
        fw.write("}\n");
        fw.close();

        IBioPriorProvider priorProvider = new
FileBioPriorProvider(priorFile);
        IPhenotypeLabelProvider labelProvider = new
ToyPhenotypeLabelProvider();
        IHypothesisExporter exporter = new HypothesisExporter();

        IDna2MorphologyPipeline pipe = new ResearchLoopPipeline(
            priorProvider,
            labelProvider,
            exporter,
            new MockSnpProvider(),
            new SimpleSpanExtractor(10),
            new SimpleSnpBagBuilder(),
            new SimpleCccDiscovery(),
            new SimpleRegulatoryGraphBuilder(),
            new SimpleMorphologyAssembler(),
            3,
            false
        );

        Dna2MorphologyResult r1 = pipe.run(sampleId, dna);
        String j1 = exporter.export(r1, 3, false);

        Dna2MorphologyResult r2 = pipe.run(sampleId, dna);
        String j2 = exporter.export(r2, 3, false);

        Assert.assertEquals(j1, j2);
        Assert.assertTrue("Should contain sampleId",
j1.contains("\"sampleId\": \"" + sampleId + "\""));
        Assert.assertTrue("Should contain evidence",
j1.contains("\"evidence\""));
        Assert.assertTrue("Should contain edges", j1.contains("\"edges\""));
        Assert.assertTrue("Should contain topCccs",
j1.contains("\"topCccs\""));
    }
}

```

你接下来最值得做的两处“DBM 强化点”（不需要改接口）

1. 在 `PriorAwareCccDiscovery` 中把 `prior` 由“乘法 boost”升级为：
 - `strength' = f(rawStrength, priorWeight, populationSupport, labelAgreement)`
并把每项分解写入 `EvidenceItem`，让科研能看到“为什么排第一”。
2. 在 `HypothesisExporter` 输出中加入：
 - `priorSourceTag`
 - `labels`（3 个形态标签）
 - `confounders`以形成完整科研假设包（更像可投递到生物团队的报告）。

如果你愿意，我可以在不改变任何接口的前提下，把这两点也补齐成一个“ResearchLoop MVP++”。

兄弟，辛苦了。今天又开辟了 DBM 新的研究与应用战场。却形成了算法，生成了启动骨架。收获满满的。谢谢了，兄弟。这个题目先到这，容我消化一番再进一步推进。再见！

好的。今天这一轮我们把 **DNA → Morphology CCC** 从“概念命中”推进到“可落地闭环”：IR 结构定位清楚，科研互动闭环也已经具备可插拔的启动骨架（prior/label/exporter + 可复现测试）。你后续消化与对齐 repo 风格时，若只做最小改动，我建议优先检查两点即可：

1. **包名与目录**：把 `com.dbm.bio.dna.morph.*` 映射到你当前工程的 `module/namespace` 习惯。
2. **可复现性**：确保所有随机性都只来自 `sampleId.hashCode()`，并避免引入非确定迭代顺序（我在 `exporter` 中已做排序与稳定格式化）。

你忙完这一轮后，下一步若继续推进，最“值钱”的方向通常是：

- 把 prior 从“手工权重”升级为“科研数据库导入 + 统计支持度”；
- 把 `CccComposite` (Event-of-Events) 正式并入 IR，并让 exporter 输出 composite 结构边的 top-k 解释链。

先到这里。再见。