Мы создадим адаптер, который будет изменять не *какие* эмоции существуют, а **как** они возникают, как долго длятся и как влияют друг на друга. Мы будем опираться на обобщенные психологические и биологические тенденции, понимая, что это — модель, а не абсолютная истина.

**Философия: Гендер как "Прошивка" (Firmware)**

Мы будем рассматривать female и male не как разные программы, а как **разные "прошивки"** для одного и того же "железа". Эта "прошивка" будет влиять на базовые параметры всей системы: гормональный фон, чувствительность к определенным стимулам и стратегии обработки эмоций.

**План Реализации: "GenderAdapter"**

Мы не будем писать два разных движка. Мы создадим **класс-адаптер**, который будет загружать разные конфигурационные профили и применять их к нашему SentioEngine.

**Шаг 1: Создание "Гендерных Профилей"**

Мы вынесем все изменяемые параметры в отдельные JSON-файлы.

**profiles/female\_profile.json:**

* Опираемся на тенденции к более высокой эмпатии, более сильной реакции на социальные стимулы и более длительному удержанию эмоциональной памяти.

codeJSON

{

"profile\_name": "Female",

"hormone\_genome": {

"libido": {"base\_level": 0.25, "decay\_rate": 0.06},

"attachment": {"base\_level": 0.45, "decay\_rate": 0.07}

},

"neuromediator\_genome": {

"oxytocin": {"base\_level": 0.6, "decay\_rate": 0.08}

},

"emotion\_modifiers": {

"доверие": {"intensity\_multiplier": 1.2, "decay\_modifier": 0.9},

"грусть": {"decay\_modifier": 0.8},

"гнев": {"decay\_modifier": 1.2}

},

"personality\_modifiers": {

"neuroticism\_bias": 1.1,

"agreeableness\_bias": 1.2

},

"memory\_config": {

"emotional\_memory\_strength": 1.3

}

}

**profiles/male\_profile.json:**

* Опираемся на тенденции к более прямой реакции на стресс (гнев/агрессия), более быстрому эмоциональному "охлаждению" и фокусу на решении проблем.

codeJSON

{

"profile\_name": "Male",

"hormone\_genome": {

"libido": {"base\_level": 0.35, "decay\_rate": 0.05},

"attachment": {"base\_level": 0.30, "decay\_rate": 0.1}

},

"neuromediator\_genome": {

"norepinephrine": {"base\_level": 0.3, "decay\_rate": 0.25}

},

"emotion\_modifiers": {

"гнев": {"intensity\_multiplier": 1.3, "decay\_modifier": 0.8},

"страх": {"decay\_modifier": 1.1},

"грусть": {"decay\_modifier": 1.3}

},

"personality\_modifiers": {

"neuroticism\_bias": 0.9,

"agreeableness\_bias": 0.8

},

"memory\_config": {

"emotional\_memory\_strength": 0.9

}

}

**Шаг 2: Создание Класса GenderAdapter**

Этот класс будет загружать профиль и "патчить" базовые определения движка.

**engine/gender\_adapter.py:**

codePython

import json

from copy import deepcopy

from .emotion\_definitions import EMOTION\_GENOME, NEUROMEDIATOR\_GENOME, HORMONE\_GENOME

class GenderAdapter:

def \_\_init\_\_(self, profile\_name: str):

self.profile\_name = profile\_name

self.profile\_data = self.\_load\_profile(profile\_name)

# Применяем модификации к базовым геномам

self.emotion\_genome = self.\_apply\_emotion\_modifiers()

self.neuromediator\_genome = self.\_apply\_neuromediator\_modifiers()

self.hormone\_genome = self.\_apply\_hormone\_modifiers()

self.memory\_strength = self.profile\_data.get("memory\_config", {}).get("emotional\_memory\_strength", 1.0)

def \_load\_profile(self, name: str):

with open(f"profiles/{name.lower()}\_profile.json", "r") as f:

return json.load(f)

def \_apply\_emotion\_modifiers(self):

modified\_genome = deepcopy(EMOTION\_GENOME)

modifiers = self.profile\_data.get("emotion\_modifiers", {})

for emotion, mods in modifiers.items():

if emotion in modified\_genome:

if "decay\_modifier" in mods:

modified\_genome[emotion]["base\_decay\_rate"] \*= mods["decay\_modifier"]

# Другие модификаторы могут быть добавлены сюда

return modified\_genome

# ... (Аналогичные методы \_apply\_neuromediator\_modifiers и \_apply\_hormone\_modifiers) ...

def \_apply\_hormone\_modifiers(self):

modified\_genome = deepcopy(HORMONE\_GENOME)

profile\_hormones = self.profile\_data.get("hormone\_genome", {})

for hormone, config in profile\_hormones.items():

if hormone in modified\_genome:

modified\_genome[hormone].update(config)

return modified\_genome

def get\_personality\_bias(self, trait\_name: str) -> float:

"""Возвращает множитель для черты личности."""

return self.profile\_data.get("personality\_modifiers", {}).get(f"{trait\_name}\_bias", 1.0)

**Шаг 3: Интеграция Адаптера в SentioEngine**

SentioEngine теперь будет инициализироваться с адаптером.

**engine/sentio\_engine.py (Ключевые изменения):**

codePython

from .gender\_adapter import GenderAdapter

class SentioEngine:

def \_\_init\_\_(self, gender\_profile: str = "female"):

self.adapter = GenderAdapter(gender\_profile)

# Используем модифицированные "геномы" из адаптера

self.emotion\_genome = self.adapter.emotion\_genome

self.neuromediator\_genome = self.adapter.neuromediator\_genome

self.hormone\_genome = self.adapter.hormone\_genome

# ... (инициализация self.active\_emotions, etc. как раньше, но с новыми геномами) ...

# Инициализируем личность с учетом гендерного смещения

self.personality = {

"neuroticism": 0.5 \* self.adapter.get\_personality\_bias("neuroticism"),

# ... другие черты ...

}

# ... (внутри методов update() и inject\_stimulus() мы теперь работаем с self.emotion\_genome и т.д.,

# которые уже настроены адаптером) ...

def \_create\_emotional\_memory(self, emotion: ActiveEmotion, keywords: list):

"""Создает "шрам" в памяти."""

# Сила "шрама" зависит от гендерного профиля

strength = emotion.intensity \* self.adapter.memory\_strength

# ... (дальнейшая логика сохранения в EmotionalMemory.json) ...

**Как Это Меняет Поведение на Практике**

Давайте рассмотрим один сценарий: **пользователь выражает грусть**.

* **Female Профиль:**
  1. **Восприятие:** Множитель для эмпатии (доверие) может быть выше, поэтому система сильнее реагирует на социальные сигналы.
  2. **Реакция:** Активируется эмоция Грусть.
  3. **Динамика:** decay\_modifier для грусти < 1.0, поэтому это чувство будет затухать **медленнее**.
  4. **Память:** emotional\_memory\_strength > 1.0. Это событие с большей вероятностью создаст сильный "якорь" в памяти, и в будущем упоминание той же темы может снова вызвать легкую грусть.
  5. **Поведение:** Длительное состояние эмпатии и поддержки.
* **Male Профиль:**
  1. **Восприятие:** Реакция может быть менее интенсивной изначально.
  2. **Реакция:** Активируется эмоция Грусть.
  3. **Динамика:** decay\_modifier для грусти > 1.0. Эмоция пройдет **быстрее**. Может активироваться Гнев (как реакция на беспомощность).
  4. **Память:** emotional\_memory\_strength < 1.0. Событие с меньшей вероятностью оставит глубокий след.
  5. **Поведение:** Быстрый переход от эмпатии к **проблемно-ориентированному** поведению ("Как мы можем это исправить?").

**Вывод**

Вводя "гендерный адаптер", мы добились невероятной гибкости. Мы не "зашиваем" поведение в код. Мы создаем **движок симуляции** и **набор профилей**. Теперь вы можете:

* **Создавать разные личности**, просто меняя gender\_profile при инициализации.
* **"Тюнинговать"** эти профили, меняя цифры в JSON, чтобы добиться нужного характера.
* В будущем создавать **не только male/female профили**, но и cautious (осторожный), adventurous (авантюрный), artistic (артистичный), каждый со своими уникальными настройками "химии" и восприятия.

Вы перешли от моделирования одной личности к созданию **фабрики по производству личностей**.