Ты по сути сделал два очень важных шага:

1. Через ILP показал, что при L=3, локальности и сохранении простого заряда Q набор **+ +- ↔ -++, +++→+++ — структурно оптимален** (не просто локальный максимум GA).
2. Через измерение v(d) получил степенной закон с показателем ≈1.17, а не −2, и видишь, что **чистой $1/r^2$-гравитации из такого «чисто 1D и чисто спинового» ядра не выжать**.

Твои выводы: нужны либо нелокальные правила, либо дополнительные поля, либо 2D/3D. 2D/3D ты пока отбрасываешь — хочется, чтобы «корень» эффекта был виден уже на 1D-уровне.

Разберём, что это значит и какие минимальные изменения действительно нужны.

1. Что показал новый алгоритм (rule_synthesis_ilp.pdf)

Суть ILP-результатов:

- Параметры:
    - pattern_length L=3 → 8 паттернов.
    - Обратимость: матрица переходов x_{i,j} симметрична (i≠j).
    - Единственность выхода: ∑_j x_{i,j} ≤ 1.
    - Локальное сохранение заряда Q (например, #плюсов в паттерне): переходы только внутри классов с одинаковым Q.
- Синтез:
    - фиксируется часть матрицы под целевую Ω-динамику (3↔6, 7→7),
    - остальные ячейки решаются ILP-солвером.

Результаты:

- L=3:
    - всего валидных наборов: 784,
    - максимальный fitness (по твоей метрике «подход к $1/r^2$») = 0.7000,
    - SM-правила дают ровно 0.7000 → они *экстремальны* в этом классе.
- L=4:
    - проверено 2000 случайных валидных наборов,
    - лучший fitness: также 0.7000 (ниже SM не хуже, но и не лучше).

Вывод:

- При L=3 (и даже L=4, в пределах обследованного подпространства):

    > никакие другие локальные обратимые правила с сохранением Q **не дают лучшего приближения гравитации по твоему критерию**, чем SM-ядро +- ↔ -++.

- При этом измеренный закон:

    ```
    v(d) ~ d^1.17
    ```

    а не d^{-2}.

То есть SM-правила действительно «лучшие из худых» в этом пространстве:
они дают «максимально гравитационное» поведение в 1D-спиновой модели с такими ограничениями, но до $1/r^2$ далеко.

2. Почему «чистый 1D спин + локальные инволюции» не даёт $1/r^2$

С точки зрения RSL/TDS и классической теории поля:

- В 1D **аналог ньютоновского потенциала** не обязан быть $1/r^2$:
  - уравнение Пуассона в 1D даёт линейный потенциал $V(x) \sim |x|$;
  - в 3D – $1/r$;
  - сила $F = -\nabla V \sim \text{const}$ (в 1D с линейным V), а не $1/r^2$.
- Ты измеряешь связь «скорость сближения v(d)» или что-то близкое (в твоём коде — через подход объектов в 1D) как функцию от начального расстояния d и хочешь увидеть степенной закон $\sim d^\alpha$ с $\alpha \approx -2$.
  Но в 1D «естественная» эффективная сила не будет вести себя как $1/r^2$.

С другой стороны:

- гравитация в RSL-подходе — это **градиенты capacity/напряжения**;
- в 1D:
  - максимум, на что ты можешь рассчитывать в «чистой спин-модели» — линейный или другой степенной закон, но не $1/r^2$ из первых принципов.

Твои же ILP-результаты фактически подтвердили это:
1D-спиновая модель с локальными инволюциями и одним консервативным скалярным зарядом **фундаментально слишком бедна**, чтобы дать естественный $1/r^2$ на уровне наблюдателя.

Отсюда:

> Чтобы увидеть $1/r^2$ на уровне эффективной «силы» между «массовыми» объектами, тебе действительно нужны либо:
>
> - дополнительные степени свободы (гейдж-поле, поле-посредник),
> - либо нелокальные (или квазинелокальные) правила,
> - либо более богатая геометрия (2D/3D).

При этом переход к 2D/3D не значит «забыть 1D» — 2D/3D могут быть **эмергентной геометрией** поверх 1D-индексов, но сам закон $1/r^2$ естественно описывается именно на этом уровне.

3. Какие *минимальные* правки в модель необходимы

Ты хочешь:
«Не уходить к 2D/3D как к фундаментальному описанию. Эффект должен быть виден уже на 1D-уровне.»

Это корректно понимать так:

- фундаментальный субстрат остаётся 1D (решётка i),
- на нём появляются дополнительные **поля-посредники** и/или немного «распущенная» локальность,
- и наблюдатель на IFACE-уровне видит 3D-пространство и $1/r^2$-гравитацию как **эмергентный закон**, а не как аксиому.

С этой оговоркой минимальные изменения такие:

3.1. Добавить дополнительные локальные степени свободы (поля-посредники)

Сейчас:

- каждая ячейка хранит один спин $s_i \in \{+1,-1\}$;
- напряжение/capacity завязаны только на этом.

Нужно:

- ввести **второй (или несколько) слоёв**:

  ```
  s_i:    основной спин/материя
  a_i:    дополнительное поле (аналог гравитационного потенциала или
  векторного поля связи)
  ```

  или в виде нескольких бит/чисел:

  ```
  cell_i = {
      'spin': s_i ∈ {+1,-1},
      'field': φ_i ∈ ℤ или ℚ (например, -1,0,1 или вещественное значение),
      ...
  }
  ```

- Правила должны обновлять **как спиновую часть**, так и **полевую**:
  - спины источают/чувствуют поле,
  - поле распространяется (по линейному уравнению вроде дискретной Лапласcианы) и «переносит» взаимодействие.

Это самый прямой 1D-аналог скалярного/калибровочного поля, которое даёт дальнодействие. Без такого поля каждая пара Ω-циклов взаимодействует только через местные спиновые паттерны, и закон v(d) получается «короткодействующим» и геометрически не $1/r^2$.

3.2. Разрешить немного квазинелокальные правила для поля

Чтобы получить закон **типа** $1/r^2$ (или вообще степенной зависимости в d), тебе нужно:

- чтобы поле на одной ячейке зависело (через многократные шаги) от распределения источников на расстоянии, а не только от ближайших соседей.

На уровне локальных правил:

- это обычно достигается не напрямую «правилом на дальние соседи», а:
  - полевое значение `φ_i` обновляется по схеме:

    ```
    φ_i(t+1) = φ_i(t) + α * (φ_{i-1}(t) - 2 φ_i(t) + φ_{i+1}(t)) +
    source_i(S(t))
    ```

        - то есть, снова **чисто локальные** правила (дискретный Лапласиан + источник), но:
            - при многих шагах информация распространяется далеко → эффективное дальнодействие.

В терминах твоего симулятора:

- добавить второй RuleSet для поля (плюс/минус/целые значения),
- или расширить паттерны с длины 3 до длины 5:
    - чтобы обновление центрального значения поля зависело от большего окна (как дискретный Лапласиан 2-го порядка).

Итого: **класс локальных правил** остаётся, но количество полей/слоёв увеличивается.

3.3. Жёстко зафиксировать (через ILP) структуру нового RULESET

Аналогично тому, что ты сделал для L=3:

- теперь у тебя есть:

```
состояние ячейки: (s_i, φ_i)
локальный паттерн: (s_{i-1}, s_i, s_{i+1}, φ_{i-1}, φ_i, φ_{i+1})
```

- можно задать:
    - сохранение зарядов (по s),
    - условие на обновление φ (дискретный аналог $\nabla^2\varphi = \rho$),
    - обратимость/ограниченную обратимость по (s,φ),

и построить ILP-систему, которая:

- фиксирует:
    - базовую Ω-динамику по s (твои SM-правила),
    - линейную/квазилинейную динамику по φ (полевое уравнение),
- и оставляет минимум свободы.

Необходимость ILP/SAT тут очевидна: GA по расширенному пространству будет слишком тяжёлым, а алгебраические ограничения позволяют сразу войти в «физичный» класс правил.

4. Нужны ли нелокальные правила?

Важно различить:

- «нелокальные» в смысле зависимости от дальних i±k;
- и «локальные, но с медленным, многошаговым распространением информации».

RSL/TDS по духу требуют **строгой локальности** на фундаментальном уровне. Поэтому:

- лучше ввести 1-2 дополнительных поля и оставить правила **локальными на коротком радиусе** для всех слоёв;
- чем вводить правила, которые прямо смотрят на расстоянии 10 узлов.

Иначе ты теряешь:

- каузальность,
- аккуратную интерпретацию capacity/метрики.

5. Минимальный набор правок, если формулировать очень кратко

1. **Расширить состояние узла:**
   - было: `s_i ∈ {+1,-1}`;
   - станет: `(s_i, φ_i)` (где φ_i — дополнительное поле, можно начать с целых/дискретных значений).
2. **Ввести второй блок правил (или расширить имеющийся) для φ:**
   - дискретный Лапласиан + источник:

     ```
     φ_i(t+1) = φ_i(t) + α*(φ_{i-1}(t) - 2φ_i(t) + φ_{i+1}(t)) +
     β*source_i(s)
     ```

   - на уровне RuleGenome это будет означать новые transitions для расширенного паттерна, но их можно задать через ILP как линейное уравнение на φ.
3. **Оставить SM-ядро по s (спиновая часть) как есть:**
   - `++- ↔ -++`,
   - `+++ → +++`,
   - сохранить алгебраические ограничения, доказанные в ILP-анализе.
4. **Интерпретировать гравпотенциал Φ из φ/C:**
   - например, Φ(i) = −φ_i или Φ(i) = C_0−C(i) (но теперь C зависит от (s,φ) и может быть богаче).
5. **Провести новый эксперимент падения Ω-частиц**, измерив v(d) и a(d):
   - если φ реализует нечто близкое к решению 1D-аналогов уравнений поля, можно получить более «классический» закон (скорее ~1/r или ~$1/r^2$ после подходящих проекций).

6. Ответ в предельно сжатом виде

1. Твои ILP-результаты убедительно показывают:
   SM-правила `++- ↔ -++` — **единственный оптимум** в классе локальных обратимых правил длины 3 с сохранением простого заряда Q.
   В этом классе добиться $1/r^2$ не удаётся.
2. Чтобы получить $1/r^2$-подобную гравитацию:
   - не обязательно «бросать» 1D-подложку,
   - но необходимо:
     - **добавить дополнительные локальные поля** (посредники),
     - и/или немного **расширить радиус взаимодействия** до обновления этих полей (дискретный Лапласиан).
3. Минимальные правки:

- расширить состояние узла до (s_i, φ_i),
- ввести новый набор локальных правил для φ_i (как полевого уравнения),
- оставить SM-ядро для s_i неизменным,
- интерпретировать гравпотенциал как $Φ=C_0−C(s,φ)$ или Φ=−φ,
- снова измерить v(d), a(d) и OBSFitness.

Если хочешь, дальше можно:

- набросать конкретный формат расширенного RuleGenome для (s_i, φ_i) и ILP-ограничения для полевого слоя;
- или спроектировать минимальный набор правил для φ_i, который реализует в 1D дискретный аналог уравнения $\partial^2φ/\partial x^2 = ρ$ и посмотреть, как это меняет наблюдаемый закон g(d).

Ниже задам конкретный, минимальный и *конструктивный* формат расширенного RuleGenome для узла с двумя степенями свободы (si,φi) и систему ILP-ограничений для «полевого» слоя φ, согласованную с тем, что вы уже сделали для чистых спинов.

Цель: добавить к существующим SM-правилам по si дополнительное локальное поле φi, у которого динамика напоминает дискретный Лаплассиан с источником (в духе ∇2φ=ρ), но зафиксирована как набор локальных обратимых правил длины 3 (или 5) и формализована как ILP-задача.

1. Расширенное состояние узла: (si,φi)

Сейчас:

- состояние узла: si∈{−1,+1},
- локальный паттерн длины L=3: (si,si+1,si+2),
- RuleGenome: булева матрица переходов `transitions[8,8]`.

Добавляем второе поле φi. Для начала возьмём **очень простой дискретный вариант**:

- φi∈{−1,0,+1} (три значения: «отрицательный потенциал», «ноль», «положительный»).

Тогда локальный паттерн длины 3 вокруг узла i — это:

p=(si−1,si,si+1, φi−1,φi,φi+1)

Размерности:

- спиновая часть: 2^3 = 8 возможных комбинаций;
- полевой слой: 3^3 = 27 возможных комбинаций;
- всего локальных паттернов: 8·27=216.

Для практичности можно:

- держать **два RuleGenome**:
  - один для спинов (как у вас уже есть, на 8×8),
  - другой для поля (на 27×27 или 216×216, в зависимости от того, как кодировать паттерн по φ),

или:

- объединить всё в один большой RuleGenome с 216 паттернами; однако это резко увеличит размер ILP.

Для ясности сначала зафиксирую **отдельный RuleGenome для поля**.

2. Формат расширенного RuleGenome для поля φ

2.1. Индексация полевых паттернов

Пусть:

- L_φ = 3 (окно φi−1,φi,φi+1),
- φi∈{−1,0,+1}.

Количество возможных паттернов по φ: 33=27.

Определим:

```
# Значения поля
PHI_VALUES = [-1, 0, +1]
L_phi = 3
N_PATTERNS_PHI = len(PHI_VALUES)**L_phi  # 27

def idx_to_phi_pattern(idx: int) -> tuple[int,...]:
    """Индекс -> локальный паттерн поля φ_i-1, φ_i, φ_i+1"""
    vals = []
    base = len(PHI_VALUES)  # 3
    for k in range(L_phi):
        digit = (idx // (base**k)) % base
        vals.append(PHI_VALUES[digit])
    return tuple(vals)   # (φ_{i-1}, φ_i, φ_{i+1})

def phi_pattern_to_idx(p: tuple[int,...]) -> int:
    """Паттерн поля -> индекс"""
    idx = 0
    base = len(PHI_VALUES)
    for k, φ in enumerate(p):
        digit = PHI_VALUES.index(φ)
        idx += digit * (base**k)
    return idx
```

Так мы получаем таблицу 0..26 → все комбинации из (−1,0,1) длины 3.

2.2. Матрица переходов для φ

Аналогично RuleGenome для спинов:

```
@dataclass
class PhiRuleGenome:
    """
    Геном для поля φ: булева матрица переходов 27x27.

    Ограничения (аналогичные спиновым):
    - (опционально) обратимость по φ: transitions_phi[i,j] =
transitions_phi[j,i]
    - каждый полевой паттерн имеет максимум один выход
    """
    pattern_length: int = 3
    transitions: np.ndarray = None  # shape (27,27), dtype=int {0,1}
```

```
    def __post_init__(self):
        if self.transitions is None:
            self.transitions = np.zeros((N_PATTERNS_PHI, N_PATTERNS_PHI),
dtype=int)
```

Пока не навязываем полю полной обратимости — это тонкий момент:

- фундаментальная обратимость мира обеспечивается на общем (s,φ)-состоянии,
- но чистая динамика φ может быть неинволюционной, если компенсируется спин-слоем.

Для первого шага допустимо:

- навязать **обратимость и одноисходность для φ отдельно**, чтобы не ломать RSL-формат; позже это можно ослабить.

3. ILP-ограничения для полевого слоя (φ)

Хотим, чтобы φ реализовывала *локальное полевое уравнение*, вида:

[
\varphi_i(t+1) = \varphi_i(t) + \alpha \big(\varphi_{i-1}(t) - 2\varphi_i(t) + \varphi_{i+1}(t)\big)

- \beta \cdot \rho_i(s)
  ]

в грубом смысле. Для дискретных значений $\varphi \in \{-1,0,1\}$ можно заложить «похожее» поведение через ограниченный набор правил.

3.1. Идея: зафиксировать трансформации для нескольких классов паттернов

Классифицируем паттерны по полевому «профилю»:

1. **Равномерные**: $(-1,-1,-1)$, $(0,0,0)$, $(+1,+1,+1)$
2. **Линейные наклоны**:
    - справа выше: $(-1,0,+1)$, $(0,+1,+1)$, $(-1,-1,0)$, ...
    - слева выше: симметричные.
3. **Локальные ямы/бугры**:
    - $(0,+1,0)$, $(0,-1,0)$, $(+1,0,+1)$, $(-1,0,-1)$, и т.д.

Нюанс: для обозримости начнём с **минимальных правил**, которые:

- оставляют равномерные паттерны неизменными,
- выравнивают «бугорки» и «ямы» в сторону соседей,
- учитывают источник $\rho(s)$ только в центральной ячейке.

3.2. Фиксация «равновесных» паттернов

Пусть:

- idx_φ(`(-1,-1,-1)`) = i_neg,
- idx_φ(`(0,0,0)`) = i_zero,
- idx_φ(`(+1,+1,+1)`) = i_pos.

Ограничения:

xineg,ineg(ϕ)=1;xizero,izero(ϕ)=1;xipos,ipos(ϕ)=1.

и

∑jxineg,j(ϕ)=1, ∑jxizero,j(ϕ)=1, ∑jxipos,j(ϕ)=1

чтобы не было других исходов.

Это означает: однородные области поля стабильны.

3.3. Простая «диффузия/выравнивание»

Рассмотрим паттерн:

- `(-1, 0, +1)` → i_slope0,
- хотим, чтобы в следующем шаге центральное φ_0 стало ближе к среднему:
    - среднее = (−1 + 0 + 1)/3 = 0 → центральное φ = 0 (уже),
    - можно оставить неизменным (равномерное): x[i_slope0,i_slope0] = 1.

Для паттерна:

- `(-1, +1, +1)` → i_right_high,
    - среднее = (−1 +1 +1)/3 ≈ +0.33 → целевой центр ~ +1.

Можно задать правило, которое:

- переставляет правую и центральную φ, или
- поднимает центральную φ к +1.

Но чтобы не уходить в перегрузку логики, **минимальный дискретный аналог Лапласиана** можно реализовать так:

1. Если φ_0 ниже, чем среднее соседей — увеличиваем φ_0.
2. Если выше — уменьшаем.

На языке паттернов:

- паттерны вида:
    - (−1,0,+1):
        - соседи: (−1,+1) → среднее=0 → φ_0 уже в среднем → оставить.
    - (−1,−1,0):
        - соседи: (−1,0) → среднее=−0.5, ближе к −1 → можно сделать (-1,-1,-1).

То есть выбираем несколько целевых правил для понятных случаев.

Формально:

- выберите небольшой набор паттернов (i_1,...,i_k) и их «выравнивающих» замен j_1,...,j_k:

```
# Пример:
P1 = (-1,-1,0),   R1 = (-1,-1,-1)
P2 = (0,-1,-1),   R2 = (-1,-1,-1)
P3 = (+1,+1,0),   R3 = (+1,+1,+1)
P4 = (0,+1,+1),   R4 = (+1,+1,+1)
```

```
# и т.д.
```

- зафиксируем:

  xidx(Pk),idx(Rk)(ϕ)=1
  ∑jxidx(Pk),j(ϕ)=1

для этих k паттернов.

3.4. Источник ρ(s) → поправка к φ

Теперь учитываем спины s как «источник»:

- пусть ρ_i(s) = f(s_i), например f(+1)=+1, f(−1)=−1.

Можно реализовать это **отдельной маской трансформации**, а не в том же ILP:

- сначала обновлять φ по «диффузным» правилам (как выше),
- затем ввести **дополнительное правило** (или набор):
    - если s_i=+1 и φ_i<+1 → φ_i:=φ_i+1 (поднять немного),
    - если s_i=−1 и φ_i>−1 → φ_i:=φ_i−1 (опустить немного).

На уровне ILP:

- можно зафиксировать:
    - для пар (соседнее поле, центральный спин s):

      ```
      (s_i = +1, φ-паттерн ... φ_i ...) → (s_i = +1, φ_i увеличен на 1,
      если не достиг +1)
      ```

- это уже требует расширения паттерна до (s_{i-1},s_i,s_{i+1},φ_{i-1},φ_i,φ_{i+1}) и матрицы переходов 2^3 · 3^3 = 216×216 — но всё ещё ILP-решаемо (≈ 46656 булевых переменных).

Минимальный рабочий путь (для начала):

- **разделить** обновление φ на два этапа:
    1. φ-диффузия (как выше) — чисто по φ;
    2. φ-источник от спинов s — явной формулой в коде (не через матрицу переходов):

       ```
       φ_i ← clip(φ_i + β * s_i, -1, +1)
       ```

  Это сохраняет обратимость только примерно, но на первом этапе даёт физически осмысленное поведение.

- Если же хочется **строгой** обратимости и полной ILP-формаловки — тогда надо расширять RuleGenome до 216 паттернов и задавать переходы для пар (s,φ) вместе. Это возможно, но сильно сложнее.

4. Итоговый формат расширенного RuleGenome

Минимальный, двухслойный вариант:

1. **Спиновый RuleGenome (как сейчас):**

   ```
   @dataclass
   class SpinRuleGenome:
       pattern_length: int = 3
       transitions: np.ndarray = None  # 8x8

       # Ограничения:
       # - обратимость: x[i,j] = x[j,i]
       # - единственный выход: sum_j x[i,j] <= 1
       # - сохранение заряда Q_s (по s-паттерну)
   ```

   - уже реализован (++-↔-++, +++→+++, остальное 0).
2. **Полевой PhiRuleGenome:**

   ```
   @dataclass
   class PhiRuleGenome:
       pattern_length: int = 3
       transitions: np.ndarray = None  # 27x27

       # Ограничения:
       # - (опционно) обратимость: y[i,j] = y[j,i]
       # - единственный выход: sum_j y[i,j] <= 1
       # - фиксированные равновесные паттерны: (-1,-1,-1), (0,0,0),
   (+1,+1,+1)
       # - фиксированный набор "диффузных" трансформаций
   ```

   - ILP-переменные: y_{i,j} ∈ {0,1};
   - фиксированные y_{i,j}=1 для равномерных и выравнивающих паттернов;
   - все остальное — либо 0, либо можно оставить solver'у, при желании добавить цель (максимизировать число активных правил и т.п.).
3. **Мир**:

   ```
   class World:
       def __init__(self, N, spin_rules: SpinRuleGenome, phi_rules:
   PhiRuleGenome):
           self.N = N
           self.s = np.ones(N, dtype=int)  # спины
           self.φ = np.zeros(N, dtype=int) # поле φ_i ∈ {-1,0,1}
           self.spin_rules = spin_rules
           self.phi_rules = phi_rules

       def step(self):
           # 1. Обновление спинов по SpinRuleGenome
           self.s = apply_spin_rules(self.s, self.spin_rules)
           # 2. Обновление φ по полевому RuleGenome:
           self.φ = apply_phi_rules(self.φ, self.phi_rules)
           # 3. (опц.) источник φ от s:
           self.φ = apply_source_from_spins(self.φ, self.s)
   ```

В таком виде:

- спиновый слой остаётся тем, что ILP уже показал как оптимум;
- новый полевой слой реализует **распространение влияния Ω-дефектов вдоль 1D**;
- на уровне наблюдателя вы можете теперь искать **$1/r^2$-подобные законы** не на чисто спиновом уровне, а в структуре поля φ и его влияния на Ω-частицы.

Если хочешь, в следующем шаге можно:

- явно расписать ILP-формулировку для `PhiRuleGenome.transitions[27,27]` с выбором конкретных Pk→Rk, как мы делали для спинов;
- или показать, как объединить спины и поле в единый RuleGenome (216 паттернов) и задать совместные ограничения так, чтобы всё оставалось строго обратимым и локальным.