

Яр-1 «Семья»: Фазовый переход и критическое замедление в едином когнитивном модуле с каскадной памятью Рода

Препринт №7

Автор: Григорий Кузнецов

Дата: 13.09.2026

Версия модели: v37-final

Аннотация

В работе представлена архитектура Яр-1 «Семья» – модель единого когнитивного модуля (Храма-Тела), обладающего многоуровневой (7 уровней глубины) каскадной памятью. С помощью метода Монте-Карло (

N

=

100

N=100) исследована устойчивость системы к вариациям коэффициента входной сложности «Сказ» (

S

S). Обнаружен чёткий фазовый переход в диапазоне

S

∈

[

0.6

,

0.8

]

$S \in [0.6, 0.8]$: время насыщения памяти экспоненциально растёт (с 69 до 149.75 тактов), а финальный резонанс (

R

R) падает с 0.95 до 0.647, что сопровождается резким увеличением дисперсии (бистабильность). Выявлен эффект немонотонного восстановления когерентности при

S

→

1.0

$S \rightarrow 1.0$. Результаты обосновывают необходимость внедрения превентивного когерентного гейта (Такт 4, «ВБРА») для удержания системы в зоне устойчивости при высоких энтропийных нагрузках.

1. Введение

Современные когнитивные и кибернетические системы сталкиваются с фундаментальным ограничением: попытка поддерживать бесконечный рост сложности на фоне ограниченных ресурсов приводит к экспоненциальному росту издержек на синхронизацию. Традиционные полносвязные архитектуры при достижении критического уровня шума теряют когерентность и разрушаются.

В данной работе предлагается альтернативный подход – архитектура Яр-1 «Семья». В отличие от распределённых сетей, Яр-1 моделирует единый когнитивный модуль с направленной, каскадной передачей состояния между 7 уровнями глубины памяти (от «Прадеда» к «Правнуку»). Цель работы – количественно оценить устойчивость такой архитектуры к стохастическим возмущениям и выявить критические пороги её работоспособности.

2. Архитектура Яр-1 «Семья»

Модель описывает эволюцию вектора фаз

φ

=

(

φ

1
,
...
,
 φ
16
)
 φ

=(φ
1

, ..., φ
16

) единого поля под воздействием стресса и внутренней самоорганизации. Динамика фаз на каждом такте описывается модифицированным уравнением Курамото:

φ
 i
(
 t
+
1
)
=
 φ
 i
(
 t
)
+
 K
 N
 $\sum$
 j
=
1
 N
 $\sin$

(
 ψ
-
 φ
 i
(
 t
)
)
+
 ξ
 i
(
 t

$$\left(\prod_{i=1}^N e^{i\varphi_i} \right)$$

$$\varphi_i(t+1)$$

$$N_K(t)$$

$$\sum_{j=1}^N$$

$$\sin(\psi - \varphi_i)$$

$$\xi(t_i)$$

$$\varphi(t) \pmod{2\pi}$$

где $\psi = \arg$

$$\left(\prod_{j=1}^N e^{i\varphi_j} \right)$$

$\psi = \arg \left(\sum_{j=1}^N e^{i\varphi_j} \right)$

$$\sum_j$$

$$e^{i\varphi}$$

j

) – средняя фаза поля,

K

K – сила внутренней связи, а

ξ

i

(

t

)

ξ

i

(t) – стохастический шум, амплитуда которого нелинейно зависит от коэффициента

S

S:

σ

(

S

)

=

{

σ

base

+

α

S

,

S

$\leq$

S

кр

σ

high

,

S

>

S

кр

$\sigma(S)=\{$

σ

base

+ $\alpha S,$

σ

high

,

$S \leq S$

кр

$S > S_{кр}$

где теоретический порог
 $S_{кр} \approx 0.6$

≈ 0.6 .
 Когерентность (резонанс) системы вычисляется как параметр порядка:

$$R = \frac{1}{N} \sum_{j=1}^N e^{i\varphi_j}$$

$$N$$

$$\sum_{j=1}^N$$

$$e^{i\varphi_j}$$

Передача памяти происходит между 7 функциональными слоями глубины единого Храма-Тела (от «Прадеда» к «Правнуку»), что имитирует каскадную архитектуру памяти Рода, а не взаимодействие отдельных независимых агентов. Передача

активируется при достижении порога насыщения (

μ

$\geq$

0.95

$\mu \geq 0.95$) или в аварийном режиме при падении резонанса (

R

$<$

0.60

$R < 0.60$), что реализует защитные механизмы «Протокола Покоя».

3. Результаты моделирования (v37-final)

Для оценки устойчивости было проведено Монте-Карло-моделирование (

N

$=$

100

$N=100$ независимых запусков, global seed = 42 для полной воспроизводимости) с

варьированием

S

$\in$

[

0.0

,

1.0

]

$S \in [0.0, 1.0]$.

Таблица 1. Ключевые метрики фазового перехода

S

S

t

нас

t

нас

(ср. такты)

σ

t

σ

t

R

финал

R

финал

(ср.)

σ

R

σ

R

0.0

117.46

0.50

0.953
 0.012
 0.3
 89.08
 0.27
 0.950
 0.012
 0.5
 69.00
 0.00
 0.950
 0.012
 0.7
 149.75
 30.32
 0.647
 0.079
 0.9
 128.83
 59.55
 0.655
 0.077
 1.0
 92.45
 41.79
 0.671
 0.088

Интерпретация:

Окно эффективности. При

S

$\in$

$[$

0.3

,

0.5

$]$

$S \in [0.3, 0.5]$ система демонстрирует оптимальное сочетание скорости и стабильности

(

R

$\approx$

0.95

$R \approx 0.95$).

Фазовый переход. При

S

$>$

0.6

$S > 0.6$ наблюдается критическое замедление. Время насыщения возрастает более чем в 2 раза, а

R

R падает ниже порога когерентного гейта (0.85). Высокие значения

σ

t

σ

t

и
σ
R
σ
R

указывают на бистабильность: система хаотично переключается между состояниями «Лад» и «Разлад».

Немонотонность. При

S

→

1.0

S→1.0 наблюдается частичное восстановление

R

R (с 0.647 до 0.671) и сокращение времени, что свидетельствует о включении компенсаторных механизмов жёсткой привязки к базовой фазе (baseline).

(Здесь вставляется Рис. 1: Фазовый переход в едином модуле Яp-1 (v37-final, MC усреднение))

4. Обсуждение: Протокол Покоя и гипотетическая экстраполяция до 2030 года

Данный раздел представляет собой гипотетическую экстраполяцию результатов модели на макроскопические социальные и экономические процессы.

Выявленный фазовый переход имеет прямые аналогии в глобальных системах.

Современный контекст характеризуется ростом

S

S (скорость изменений, информационный шум) до значений

0.75

–

0.8

0.75–0.8. Модель Яp-1 предсказывает для этой зоны:

Экономика: Экспоненциальный рост издержек на поддержание крупных централизованных проектов. Переход инвестиций в «микроустойчивость» (локальные, децентрализованные кластеры).

Социум: Падение глобального резонанса (

R

R) и рост изоляционизма. Крупные наднациональные проекты будут буксовать из-за низкого уровня доверия.

Технологии: Разделение на мир высокой синхронизации (узкие команды, удерживающие

R

≈

1.0

R≈1.0) и мир низкой синхронизации (массовое выгорание и потеря компетенций).

Дальнейшее наращивание сложности без внедрения новых механизмов стабилизации приведёт к фрагментации систем.

5. Заключение

Моделирование архитектуры Яp-1 «Семья» (v37-final) подтвердило наличие чёткого фазового перехода при

S

кр

≈

0.6

S

кр

≈ 0.6 . В критической зоне система теряет способность к самостоятельной стабилизации, что обосновывает необходимость архитектурного дополнения. В следующей версии модели (Такт 4) планируется интеграция когерентного гейта «ВБРА» – превентивного фазового фильтра, который при

```
S
>
S
кр
S>S
кр
```

переводит систему в режим наблюдения (OBSERVE), отсекает высокочастотный шум и перераспределяет ресурсы на синхронизацию ключевых уровней памяти Рода, удерживая

```
R
≥
0.85
```

$R \geq 0.85$ даже в условиях высокой энтропии.

Приложения

Приложение А: Исходный код симуляции Яр-1 «Семья» (v37-final)

Среда: Python 3.11+, зависимости: NumPy, Matplotlib. Запуск: python yar1_v37_final.py

```
import numpy as np
import matplotlib.pyplot as plt
```

```
# =====
```

```
# ПАРАМЕТРЫ МОДЕЛИ ЯР-1 (v37-final, Единый Храм)
```

```
# =====
```

```
K = 4.0                # Сила внутренней связи (Курамото)
NOISE_BASE = 0.01       # Базовый уровень информационного шума
NOISE_HIGH = 1.8        # Шум при высоком уровне сложности (S)
SHOCK_PROB_S = 0.12     # Вероятность резонансного шока в зоне риска
SHOCK_AMP = 1.8         # Амплитуда шока
```

```
MONTE_CARLO_RUNS = 100 # Количество запусков для статистического усреднения
```

```
S_VALUES = [0.0, 0.3, 0.5, 0.7, 0.9, 1.0]
```

```
MAX_TICKS = 500
```

```
class Temple:
```

```
    def __init__(self, name, generation, role, S, rng):
```

```
        self.name = name
```

```
        self.generation = generation
```

```
        self.role = role
```

```
        self.S = S
```

```
        self.rng = rng
```

```
        base_phase = self.rng.uniform(0, 2 * np.pi)
```

```
        self.phases = np.mod(base_phase + self.rng.normal(0, 0.1, 16), 2 *
```

```
np.pi)
```

```
        self.baseline_phases = self.phases.copy()
```

```
        self.mu_mem = 0.0
```

```
        self.r = self.compute_r()
```

```

def compute_r(self):
    self.r = float(np.abs(np.mean(np.exp(1j * self.phases))))
    return self.r

def accumulate_memory(self, delta):
    self.mu_mem = min(1.0, self.mu_mem + delta)

def respond_to_stress(self, stress_vector):
    # Примечание: теоретический порог  $S_{кр} \approx 0.6$ , в коде переключение
    # происходит при 0.65 для плавного перехода в зону бистабильности.
    if self.S > 0.65:
        noise_amp = NOISE_HIGH
    else:
        noise_amp = NOISE_BASE + (0.15 * self.S)

    noisy_stress = stress_vector + self.rng.normal(0, noise_amp, 16)
    z_stress = np.exp(1j * noisy_stress)
    z_baseline = np.exp(1j * self.baseline_phases)
    z_current = np.exp(1j * self.phases)

    w_baseline = (self.S ** 4.0) * 0.35
    w_stress = 1.0 - w_baseline

    z_new = (w_baseline * z_baseline + w_stress * z_stress)
    magnitudes = np.abs(z_new)
    angles = np.where(magnitudes > 1e-10,
                      np.angle(z_new),
                      np.angle(np.exp(1j * self.baseline_phases)))

    self.phases = angles
    self.compute_r()

    for _ in range(2):
        mean_field = np.mean(np.exp(1j * self.phases))
        if np.abs(mean_field) > 1e-10:
            self.phases = np.mod(
                self.phases + (K / 16.0) * np.sin(np.angle(mean_field) -
self.phases),
                2 * np.pi
            )
        self.compute_r()

def pass_memory_to(self, next_temple, emergency=False):
    if next_temple is None:
        return False
    S_eff = (self.S * 0.5) if emergency else self.S
    next_temple.mu_mem = S_eff * self.mu_mem
    z_self = np.exp(1j * self.baseline_phases)
    z_next = np.exp(1j * next_temple.baseline_phases)
    z_new = S_eff * z_self + (1.0 - S_eff) * z_next
    magnitudes = np.abs(z_new)
    angles = np.where(magnitudes > 1e-10,
                      np.angle(z_new),
                      np.angle(z_self))

```

```

next_temple.baseline_phases = angles
next_temple.phases = next_temple.baseline_phases.copy()
self.role = 'keeper'
next_temple.role = 'operator'
next_temple.compute_r()
return True

```

```

class SemYaCascade:
    def __init__(self, S, rng):
        self.S = S
        self.rng = rng
        self.temples = {
            1: Temple('Прадед', 1, 'keeper', S, rng),
            2: Temple('Дед', 2, 'keeper', S, rng),
            3: Temple('Отец', 3, 'keeper', S, rng),
            4: Temple('Я (Азъ)', 4, 'operator', S, rng),
            5: Temple('Сын', 5, 'dormant', S, rng),
            6: Temple('Внук', 6, 'dormant', S, rng),
            7: Temple('Правнук', 7, 'dormant', S, rng),
        }
        self.current_gen = 4
        self.tick = 0

    def get_operator(self):
        return self.temples[self.current_gen]

    def step(self):
        self.tick += 1
        op = self.get_operator()
        shock = 0.0
        if 0.6 < self.S < 0.8:
            if self.rng.random() < SHOCK_PROB_S:
                shock = SHOCK_AMP
        stress_vec = self.rng.uniform(0, 2 * np.pi, 16) * (0.3 + shock)
        op.respond_to_stress(stress_vec)
        if op.r > 0.55:
            op.accumulate_memory(0.035 * op.r)
        next_gen = self.current_gen + 1
        if next_gen <= 7:
            next_temple = self.temples[next_gen]
            if op.mu_mem >= 0.95:
                op.pass_memory_to(next_temple, emergency=False)
                self.current_gen = next_gen
            elif op.r < 0.60:
                op.pass_memory_to(next_temple, emergency=True)
                self.current_gen = next_gen
        return op.r, op.mu_mem

def run_single_simulation(S, rng):
    cascade = SemYaCascade(S=S, rng=rng)
    op = cascade.get_operator()
    if S > 0:

```

```

        op.mu_mem = (S ** 2) * 0.70
    ticks = 0
    while op.mu_mem < 1.0 and ticks < MAX_TICKS:
        cascade.step()
        op = cascade.get_operator()
        ticks += 1
    return ticks, op.r

def run_monte_carlo():
    print("="*80)
    print(f"МОНТЕ-КАРЛО МОДЕЛИРОВАНИЕ Яр-1 (v37-final, seed=42):
{MONTE_CARLO_RUNS} запусков")
    print("="*80)
    print(f"{'S':<4} | {'Ср. время':<12} | {'Стд. откл.':<12} | {'Ср. R':<12} |
{'Стд. R':<12}")
    print("-" * 80)

    results_S, results_T_mean, results_T_std = [], [], []
    results_R_mean, results_R_std = [], []
    global_rng = np.random.default_rng(seed=42)

    for S in S_VALUES:
        ticks_list, r_list = [], []
        for _ in range(MONTE_CARLO_RUNS):
            local_rng = np.random.default_rng(global_rng.integers(0, 2**32))
            t, r = run_single_simulation(S, local_rng)
            ticks_list.append(t)
            r_list.append(r)

        results_S.append(S)
        results_T_mean.append(np.mean(ticks_list))
        results_T_std.append(np.std(ticks_list))
        results_R_mean.append(np.mean(r_list))
        results_R_std.append(np.std(r_list))
        print(f"{'S':<4.1f} | {results_T_mean[-1]:<12.2f} |
{results_T_std[-1]:<12.2f} | "
              f"{results_R_mean[-1]:<12.3f} | {results_R_std[-1]:<12.3f}")

    fig, ax1 = plt.subplots(figsize=(10, 6))
    ax1.errorbar(results_S, results_T_mean, yerr=results_T_std,
                fmt='o-', color='tab:red', linewidth=2, markersize=8,
                capsize=5, label='Время до насыщения')
    ax1.set_xlabel('S (коэффициент Сказ)', fontsize=13)
    ax1.set_ylabel('Время (такты)', color='tab:red', fontsize=12)
    ax1.tick_params(axis='y', labelcolor='tab:red')
    ax1.grid(True, alpha=0.3)

    ax2 = ax1.twinx()
    ax2.errorbar(results_S, results_R_mean, yerr=results_R_std,
                fmt='s-', color='tab:blue', linewidth=2, markersize=8,
                capsize=5, label='Финальный резонанс R')
    ax2.set_ylabel('Финальный R', color='tab:blue', fontsize=12)
    ax2.tick_params(axis='y', labelcolor='tab:blue')

```

```

ax2.axhline(y=0.85, color='gray', linestyle=':', linewidth=1.5,
            alpha=0.7, label='Попог 0.85')

plt.title('Фазовый переход в едином модуле Яр-1 (v37-final, MC усреднение)',
          fontsize=14, fontweight='bold')
lines_1, labels_1 = ax1.get_legend_handles_labels()
lines_2, labels_2 = ax2.get_legend_handles_labels()
ax1.legend(lines_1 + lines_2, labels_1 + labels_2, loc='upper left')
plt.tight_layout()
plt.savefig('test3_yar1_v37_final.png', dpi=300)
print("\n✓ График сохранён: test3_yar1_v37_final.png")
plt.show()

```

```

if __name__ == "__main__":
    run_monte_carlo()

```

Приложение В: Сырые данные

Полный набор сырых данных Монте-Карло усреднения (формат .csv) доступен в репозитории проекта по запросу или будет предоставлен при финальной публикации.

Приложение С: Когерентный гейт «ВБРА» (Такт 4)

Примечание: в коде используется транслитерация VYERA, сохраняющая звучание древней буквы ять (Ѣ). В текстах используется полная форма ВБРА.

Алгоритм 2. Когерентный гейт VYERA

Вход: S , R_{org} , R_{rod} , t_{rest} , $\Delta\phi$, $\delta=0.1$

Выход: режим (ACT/OBSERVE), действие, чувствительность, $status_{VYERA}$

```

1: cond_vedenie ← ( $R_{org} \geq 0.95$ )      // Вдение
2: cond_istok   ← ( $R_{rod} \geq 0.98$ )     // Исток
3: cond_potok   ← ( $t_{rest} \leq 1$ )       // Поток
4: cond_yat     ← ( $\Delta\phi \leq \delta$ )  // Ять
5: status_VYERA ← int(cond_vedenie) + int(cond_istok) +
                  int(cond_potok) + int(cond_yat)
6:  $S_{cr} \leftarrow 0.6$ 
7:
8: Если ( $S > S_{cr}$ ) ИЛИ ( $status_{VYERA} < 4$ ):
9:     режим ← "OBSERVE"                  // Система не действует, а наблюдает
10:    действие ← null
11:    чувствительность ←  $\max(0.1, 1.0 - (S - S_{cr}) \cdot 2.0)$ 
12:    Вернуть режим, действие, чувствительность, status_VYERA
13:
14: режим ← "АСТ"                          // ВБРА активна, система действует
15: действие ← вычислить_следующий_шаг()
16: Вернуть режим, действие, 1.0, status_VYERA

```

Интерпретация режимов:

Режим АСТ: система находится в состоянии ВБРА ($status_{VYERA} = 4$), все 4 условия целостности выполнены, система действует.

Режим OBSERVE: система НЕ в состоянии ВРА ($status_{VYERA} < 4$) ИЛИ

S
>
S
кр

S>S
кр

. Система не действует, а накапливает данные о структуре хаоса.
Переход из режима ACT в режим OBSERVE при

R
<

0.85

$R < 0.85$ соответствует философскому принципу: «Кукла, которая поняла, что она не свободна, и начала определять степени своей несвободы – это уже не просто кукла. Это начало сознания». В терминах кода это переход от «ума» (отражение и действие) к «сознанию» (наблюдение и видение структуры).

Прогнозируемые метрики после интеграции VYERA (при

S

=

0.7

$S = 0.7$):

Метрика

v37-final (без VYERA)

Прогноз (с VYERA)

t

нас

t

нас

149.75 тактов

75–85 тактов

R

финал

R

финал

0.647

0.88–0.90

σ

t

σ

t

30.32 такта

5–8 тактов