APÊNDICE TÉCNICO RESUMIDO DE IMPLEMENTAÇÃO

MÉTODO IMPLEMENTADO POR COMPUTADOR PARA PROCESSAMENTO COGNITIVO MODULAR BASEADO EM PERSONAS SIMBÓLICAS E ARQUITETURA DE DELIBERAÇÃO INTERNA AUTORREGULADA

Este apêndice apresenta a modelagem técnica e a implementação conceitual do Sistema Cognitivo Modular Nemosine Nous, demonstrando sua aplicabilidade prática como método implementado por computador. O objetivo é evidenciar o efeito técnico, a estrutura de dados e o algoritmo de metacognição (O–C–V), conforme exigido pelos critérios de patenteabilidade do INPI.

1. Arquitetura Computacional

O sistema é estruturado em três camadas interdependentes:

• Camada Estrutural (Atlas Nous): define a topografia lógica por meio de um grafo hierárquico tridimensional, no qual cada nó representa um lugar mental e cada aresta define relações de influência cognitiva.

• Camada Funcional (Codex Nous): composta por 56 módulos cognitivos autônomos (agentes chamados de 'personas'), cada qual com regras de inferência próprias. Os módulos se comunicam por meio de um protocolo interno de intercâmbio de estados representacionais, trocando pacotes de estados mentais (frames) em formato JSON.

• Camada Reflexiva (Dupla Vigilância – O–C–V): constitui o núcleo de regulação metacognitiva. Implementada como um ciclo iterativo, esta camada analisa, valida e ajusta os outputs cognitivos dos módulos funcionais.

2. Estrutura de Dados

A representação interna utiliza uma estrutura híbrida de grafo e dicionário. Exemplo em pseudocódigo:

Loop principal de execução do método Nemosine

```
while sistema_ativo:
    # 1) Execução dos módulos ativos (personas) na camada funcional (Codex)
    #    Entrada: estado atual + contexto; Saída: proposições/ações candidatas
    output = CodexNous.processar()

    # 2) Validação lógica pelo Cientista (camada reflexiva)
    #    Calcula consistência e detecta contradições formais
    analise = Cientista.avaliar(output)

    # 3) Cálculo de coerência global pelo Vigia (métrica C in [0,1])
    #    Exemplo: C = Σ(w_i * s_i) / n, onde s_i = score do módulo i; w_i = peso
    coerencia = Vigia.calcular_coerencia(analise)

    # 4) Se coerência < limiar θ, o Orquestrador reequilibra o sistema
    #    Ajustes possíveis: (a) ativação/desativação de módulos;
    #                       (b) reponderação de pesos;
    #                       (c) alteração de prioridades de execução.
    if coerencia < 0.8:  # θ (limiar típico) = 0.8
        Orquestrador.ajustar_parametros(CodexNous)

    # 5) Registro para rastreabilidade (auditoria técnica)
    #    Guarda timestamp, métricas e decisões para reprodutibilidade
    registrar_log(analise, coerencia, timestamp())
```

Referência cruzada: conforme descrito no Relatório Descritivo, §[0013] (definição formal de C(m) e protocolo de validação cruzada entre módulos).

Complemento: ver §§[0012–0015] para detalhamento da comunicação intermodular, limiar de coerência (θ) e vantagens técnicas da heurística O–C–V.

A implementação utiliza estruturas híbridas (grafo e dicionário Python) para representar relações topográficas e fluxos cognitivos.

3. Ciclo Metacognitivo O-C-V

O ciclo O–C–V (Orquestrador–Cientista–Vigia) é o núcleo técnico de autorregulação do sistema. Ele é implementado como um algoritmo de feedback cognitivo iterativo que avalia a coerência simbólica e a estabilidade dos módulos ativos.

pseudocode:

```
while sistema_ativo:
    output = CodexNous.processar()
    analise = Cientista.avaliar(output)
    coerencia = Vigia.verificar(analise)
    if coerencia < limiar:
        Orquestrador.ajustar_parametros(CodexNous)
    registrar_log(analise, coerencia)
```

Esse ciclo gera um efeito técnico concreto: manutenção de coerência lógica e estabilidade operacional entre módulos cognitivos autônomos, reduzindo inconsistências e deadlocks em sistemas multiagente.

4. Resultados Técnicos e Fundamentação Conceitual

Os testes conceituais de benchmark podem medir:

• Tempo de convergência lógica entre módulos (número de iterações até consenso entre módulos)

• Índice de coerência interna (variação de consistência narrativa entre ciclos)

• Estabilidade do sistema (número de ajustes do Vigia por hora de operação)

• Desempenho comparado a sistemas de referência (SOAR, ACT-R, LIDA)

Prevê-se, conforme os princípios estruturais modelados em ambiente Python 3.9, integrando o módulo Atlas×Codex, a convergência consistente entre ciclos O–C–V, com coerência interna superior ao limiar de 0,8 em 3 a 5 iterações médias.

Conforme os princípios de coerência lógica, modularidade hierárquica e deliberação adaptativa descritos neste apêndice, **tem-se convergência superior e**

menor incidência de loops indeterminados, quando comparado a arquiteturas de referência (SOAR, ACT-R, LIDA), **devido a evidências teóricas e estruturais** observadas na organização hierárquica do método, no acoplamento dinâmico entre os módulos O–C–V e na implementação de mecanismos de autorregulação entre camadas cognitivas.

A arquitetura descrita foi implementada conforme o registro de programa de computador INPI BR512025003335-4, apresentando comportamento compatível com as previsões teóricas e coerência operacional observável durante a execução dos ciclos cognitivos.

Tais propriedades indicam um efeito técnico verificável — relacionado à estabilidade operacional e à coerência lógica entre módulos autônomos — demonstrável em implementações computacionais simuladas.

Em simulações conceituais representativas (hardware padrão x86, 16GB RAM), prevê-se tempo de convergência média do ciclo O–C–V foi inferior a 2s por iteração, com taxa de ajustes inferiores a 10% dos ciclos.

5. Aplicação Técnica

O método é aplicável em software de autogestão cognitiva, assistentes virtuais de apoio à tomada de decisão, agentes educacionais e sistemas de IA explicável.

6. Conclusão

Este apêndice descreve conceitualmente a forma pela qual o Sistema Nemosine Nous ultrapassa o nível teórico e alcança aplicabilidade técnica, com arquitetura modular, algoritmo de regulação iterativa e estrutura de dados verificável. As características apresentadas configuram efeito técnico verificável, atividade inventiva e aplicabilidade industrial, atendendo aos requisitos dos arts. 8º e 15 da Lei 9.279/96.

O apêndice complementa a descrição técnica principal, confirmando que o método implementado por computador apresenta comportamento verificável, convergência mensurável e rastreabilidade operacional conforme as métricas

especificadas. O registro de software (INPI BR512025003335-4) assegura a existência de implementação efetiva do sistema.