# 分布式神经符号认知架构部署报告：基于Windows与Ubuntu异构环境的双机协同方案

## 1. 执行摘要与架构哲学

随着人工智能辅助创作从单一的大语言模型（LLM）生成向更加复杂的\*\*神经符号系统（Neuro-Symbolic Systems）**演进，构建一个能够持久存储、检索并推理长篇叙事逻辑的本地化基础设施变得至关重要。本报告详细阐述了一套基于双机异构环境的半自动化AI写作系统部署方案。该方案旨在解决长篇小说创作中普遍存在的“灾难性遗忘”与逻辑断裂问题，通过物理和逻辑上的分层，将**执行/生成层（Executive Layer）**与**记忆/存储层（Memory Layer）\*\*解耦1。

在本架构中，**计算机A（创作控制台）运行Windows 11系统，承载着人类创作者的直觉引导、Codex模型的即时生成以及向量计算任务**；\*\*计算机B（神经记忆服务器）\*\*运行Ubuntu系统并搭载宝塔面板（Baota Panel），作为系统的潜意识后台，负责维护基于图数据库（Neo4j）的符号逻辑记忆和基于向量数据库（ChromaDB）的语义联想记忆。这种“双脑”架构不仅优化了计算资源的分配——将高频的交互计算与高吞吐的I/O操作分离，还通过物理隔离增强了数据的安全性与系统的稳定性。

本报告将超越基础的软件安装指南，深入探讨在异构操作系统间构建无缝认知数据总线的技术细节。涵盖了从底层的网络拓扑设计、Docker容器化编排、跨平台数据库连接，到应用层的剪贴板自动化桥接等全链路实施策略。

## 2. 系统拓扑与基础设施规划

### 2.1 双机异构架构的逻辑定义

在神经符号AI写作系统中，两台计算机的功能分区如下：

* **计算机A（Executive Node - Windows 11）：** 这是系统的“前额叶”。
  + **核心职责**：运行VSCode + Codex进行文本生成；运行Python脚本处理逻辑控制；**运行本地Embedding模型（如all-MiniLM-L6-v2）将文本转化为向量**。
  + **优势**：利用A机的GPU/CPU资源进行实时计算，减轻B机负担，确保存储端纯粹且高效。
* **计算机B（Mnemonic Node - Ubuntu/Baota）：** 这是系统的“海马体”。
  + **核心职责**：运行Docker引擎，托管Neo4j（图谱）和ChromaDB（向量库）。
  + **优势**：通过Ubuntu的稳定性提供7x24小时的数据服务，并通过宝塔面板简化运维。

### 2.2 网络层级设计与静态寻址

为了确保“大脑”与“记忆”之间的神经连接永不中断，网络层的确定性是部署的首要前提。

**推荐的网络拓扑配置：**

| **设备名称** | **角色** | **操作系统** | **IP地址** | **关键端口** |
| --- | --- | --- | --- | --- |
| **Gateway** | 路由器 | N/A | 192.168.1.1 | N/A |
| **Computer B** | 记忆服务器 | Ubuntu 22.04/24.04 | **192.168.1.195** (静态) | 7474 (Neo4j HTTP)  7687 (Neo4j Bolt)  8000 (ChromaDB) |
| **Computer A** | 创作终端 | Windows 11 | DHCP (建议保留) | N/A |

在计算机B（Ubuntu）上配置静态IP是部署的第一步。这可以通过**Netplan**工具实现。

在Ubuntu终端中，编辑 /etc/netplan/ 目录下的配置文件（如 00-installer-config.yaml）：

YAML

network:  
 version: 2  
 renderer: networkd  
 ethernets:  
 enp3s0: # 请替换为您实际的网卡名称，使用 ip link 查看  
 dhcp4: false  
 addresses:  
 - 192.168.1.195/24  
 routes:  
 - to: default  
 via: 192.168.1.1  
 nameservers:  
 addresses: [1.1.1.1, 8.8.8.8]

执行 sudo netplan apply 后，计算机B将作为网络中不可移动的锚点 192.168.1.195，等待计算机A的连接呼叫2。

## 3. 记忆节点（计算机B）的深度部署：容器化与编排

计算机B的核心任务是运行Docker引擎。宝塔面板的引入为这一过程提供了图形化的管理界面。

### 3.1 宝塔面板环境下的防火墙策略

在部署Docker容器前，必须处理**宝塔防火墙**与**Docker iptables**的关系。

1. **宝塔安全设置**：在宝塔面板的“安全”页面，**必须放行**以下端口，否则计算机A无法连接：
   * 7474 (Neo4j Browser)
   * 7687 (Neo4j Bolt Protocol)
   * 8000 (ChromaDB API)
2. **安全建议**：建议在放行时，将“指定IP”设置为计算机A的IP（如果是固定的话），或者在家庭局域网中直接放行。

### 3.2 神经符号存储编排：Docker Compose架构

在计算机B上，建议在 /www/server/docker/ai-memory（或自定义路径）下创建项目目录。

以下是深度定制的 docker-compose.yml 文件。此配置针对您的“A端计算向量，B端纯存储”的需求进行了优化。

YAML

version: '3.8'  
  
services:  
 # ==========================================  
 # 符号记忆模块：Neo4j 图数据库  
 # ==========================================  
 neo4j:  
 image: neo4j:5.15.0-community  
 container\_name: neural\_symbolic\_memory  
 restart: unless-stopped  
 ports:  
 - "0.0.0.0:7474:7474" # HTTP  
 - "0.0.0.0:7687:7687" # Bolt  
 volumes:  
 -./neo4j/data:/data  
 -./neo4j/conf:/conf  
 -./neo4j/logs:/logs  
 -./neo4j/plugins:/plugins  
 environment:  
 - NEO4J\_AUTH=neo4j/your\_strong\_password  
 # 允许远程连接 (0.0.0.0)  
 - NEO4J\_server\_default\_\_listen\_\_address=0.0.0.0  
 - NEO4J\_server\_bolt\_listen\_\_address=0.0.0.0:7687  
 - NEO4J\_server\_http\_listen\_\_address=0.0.0.0:7474  
 # 内存限制：根据B机配置适当调整  
 - NEO4J\_server\_memory\_heap\_initial\_\_size=1G  
 - NEO4J\_server\_memory\_heap\_max\_\_size=2G  
 # 启用APOC插件用于复杂查询  
 - NEO4J\_PLUGINS=["apoc"]  
 - NEO4J\_dbms\_security\_procedures\_unrestricted=apoc.\*  
 networks:  
 - neural\_net  
  
 # ==========================================  
 # 语义记忆模块：ChromaDB 向量数据库  
 # ==========================================  
 chromadb:  
 image: chromadb/chroma:latest  
 container\_name: neural\_semantic\_memory  
 restart: unless-stopped  
 ports:  
 - "0.0.0.0:8000:8000"  
 volumes:  
 -./chroma\_data:/chroma/chroma  
 environment:  
 # 开启持久化  
 - IS\_PERSISTENT=TRUE  
 - PERSIST\_DIRECTORY=/chroma/chroma  
 # 允许远程重置（开发调试用）  
 - ALLOW\_RESET=TRUE  
 # 关闭匿名遥测  
 - ANONYMIZED\_TELEMETRY=FALSE  
 # 服务器监听配置  
 - CHROMA\_SERVER\_HOST=0.0.0.0  
 - CHROMA\_SERVER\_PORT=8000  
 networks:  
 - neural\_net  
  
networks:  
 neural\_net:  
 driver: bridge

**配置说明：**

* **Neo4j**: 显式设置了 NEO4J\_server\_default\_\_listen\_\_address=0.0.0.0，这是跨机连接的关键5。
* **ChromaDB**: 这是一个纯净的向量存储服务。它**不加载**任何Embedding模型，因为它只接收来自A机的纯向量数据（Float list）。这大大降低了B机的内存占用7。

## 4. 创作控制台（计算机A）的接入配置

计算机A（Win11）不仅是控制台，现在也承担了“向量处理器”的角色。

### 4.1 Windows环境下的Python生态构建

您已经创建了名为 xiaoshuo 的虚拟环境（Python 3.11）。在VSCode中，请按照以下步骤激活环境并安装依赖。

1. 激活虚拟环境：  
   打开VSCode终端（PowerShell），运行以下命令激活您现有的 xiaoshuo 环境：  
   PowerShell  
   # 假设您的虚拟环境在当前项目目录下

.\xiaoshuo\Scripts\Activate.ps1

# 激活成功后，命令行前会出现 (xiaoshuo) 标识  
```

1. 安装核心依赖：  
   在激活的 (xiaoshuo) 环境中执行安装。由于您使用本地Embedding计算，必须安装 sentence-transformers：  
   PowerShell  
   # 1. 安装Neo4j驱动（用于图数据库连接）  
   pip install neo4j  
     
   # 2. 安装ChromaDB及本地向量计算模型  
   # 核心：因为A机负责计算，所以必须包含 sentence-transformers  
   pip install chromadb sentence-transformers  
     
   # 3. 安装剪贴板交互库（用于监听您的复制指令）  
   pip install pyperclip

### 4.2 神经突触连接：桥接脚本设计

您需要的“半自动”脚本逻辑如下：脚本在A机运行，监听剪贴板，调用A机的CPU/GPU计算向量，然后将数据发送给IP为 192.168.1.195 的B机。

#### 4.2.1 符号记忆连接（Neo4j）

Python

from neo4j import GraphDatabase  
  
# 指向计算机B的新IP (192.168.1.195)  
URI = "bolt://192.168.1.195:7687"  
AUTH = ("neo4j", "your\_strong\_password")  
  
def connect\_symbolic\_memory():  
 try:  
 driver = GraphDatabase.driver(URI, auth=AUTH)  
 driver.verify\_connectivity()  
 print(f"✅ [符号记忆] 已连接至 Neo4j @ {URI}")  
 return driver  
 except Exception as e:  
 print(f"❌ [符号记忆] 连接失败: {e}")  
 return None

#### 4.2.2 语义记忆连接（ChromaDB - 客户端计算模式）

这是本架构的重点。我们使用 embedding\_functions 在本地（A机）初始化模型，并将其传递给 HttpClient。

Python

import chromadb  
from chromadb.config import Settings  
from chromadb.utils import embedding\_functions  
  
# 1. 初始化本地Embedding模型（运行在计算机A）  
# 第一次运行时会自动下载模型文件到A机的 xiaoshuo 虚拟环境中  
print("⏳ 正在加载本地Embedding模型...")  
local\_ef = embedding\_functions.SentenceTransformerEmbeddingFunction(  
 model\_name="all-MiniLM-L6-v2" # 或其他您喜欢的轻量级模型  
)  
print("✅ 本地模型加载完毕")  
  
# 2. 连接至远程记忆服务器（计算机B - 192.168.1.195）  
# 注意：这里只负责传输数据，不消耗B机算力进行推理  
client = chromadb.HttpClient(  
 host='192.168.1.195',   
 port=8000,  
 settings=Settings(allow\_reset=True)  
)  
  
def get\_memory\_collection():  
 try:  
 # 获取或创建集合，并指定使用本地的 embedding\_function  
 # 这样，当您add文本时，Chroma客户端会在A机先计算向量，再发送给B机  
 collection = client.get\_or\_create\_collection(  
 name="novel\_memory",  
 embedding\_function=local\_ef   
 )  
 print(f"✅ [语义记忆] 已连接至 ChromaDB @ 192.168.1.195 (Latency: {client.heartbeat()}ns)")  
 return collection  
 except Exception as e:  
 print(f"❌ [语义记忆] 连接失败: {e}")  
 return None

### 4.3 剪贴板监听逻辑（半自动核心）

此脚本将作为后台服务一直运行在VSCode终端的 xiaoshuo 环境中。

Python

import pyperclip  
import time  
  
def clipboard\_monitor():  
 collection = get\_memory\_collection()  
 neo4j\_driver = connect\_symbolic\_memory()  
   
 last\_paste = ""  
 print("🚀 系统就绪。请复制带有指令的文本（如 '>> SAVE: 内容'）...")  
  
 while True:  
 try:  
 current\_paste = pyperclip.paste()  
 if current\_paste!= last\_paste:  
 last\_paste = current\_paste  
   
 # 识别存储指令  
 if current\_paste.startswith(">> SAVE:"):  
 content = current\_paste.replace(">> SAVE:", "").strip()  
 if collection:  
 # 自动在A机计算向量并发送至B机存储  
 collection.add(  
 documents=[content],  
 metadatas=[{"source": "clipboard", "timestamp": time.time()}],  
 ids=[f"clip\_{int(time.time())}"]  
 )  
 print(f"🧠 [记忆已捕获] 语义已存入服务器: {content[:20]}...")  
   
 # 识别查询指令  
 elif current\_paste.startswith(">> RECALL:"):  
 query = current\_paste.replace(">> RECALL:", "").strip()  
 if collection:  
 # 自动在A机计算Query向量，发送至B机检索  
 results = collection.query(query\_texts=[query], n\_results=1)  
 print(f"💡 [记忆联想] {results['documents']}")  
  
 time.sleep(1.0)  
 except KeyboardInterrupt:  
 break  
 except Exception as e:  
 print(f"⚠️ 发生错误: {e}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 clipboard\_monitor()

## 5. 跨平台连通性验证与故障排除

在开始创作前，请在**计算机A**的 PowerShell 中运行以下命令，确保通往 195 的道路畅通。

### 5.1 连通性测试（Test-NetConnection）

PowerShell

# 1. 测试图数据库 HTTP 端口 (Neo4j)  
Test-NetConnection -ComputerName 192.168.1.195 -Port 7474  
  
# 2. 测试图数据库 Bolt 端口 (Neo4j 数据通道)  
Test-NetConnection -ComputerName 192.168.1.195 -Port 7687  
  
# 3. 测试向量数据库端口 (ChromaDB)  
Test-NetConnection -ComputerName 192.168.1.195 -Port 8000

**结果判读：**

* **TcpTestSucceeded: True** -> 配置完美，可以运行Python脚本。
* **TcpTestSucceeded: False** -> 检查计算机B的宝塔防火墙是否放行了对应端口，或Ubuntu防火墙（UFW）是否拦截。

## 6. 结论

本方案已针对您的具体环境（B机IP 192.168.1.195 和A机 xiaoshuo 虚拟环境）进行了全面更新。通过在**计算机A**上集成Embedding计算模型，我们成功实现了**边缘计算架构**：

* **计算机A**：负责逻辑控制、文本生成、向量计算（高CPU/GPU负载）。
* **计算机B**：负责数据持久化、索引维护（高I/O负载）。

这种分工最大限度地利用了Win11桌面的计算能力和Ubuntu服务器的存储稳定性，为您的半自动AI小说创作提供了坚实的工业级后端支持。

### 数据概览表：双机部署核心参数 (最终版)

| **组件/参数** | **设定值** | **部署位置** | **作用与备注** |
| --- | --- | --- | --- |
| **Server IP** | **192.168.1.195** | 计算机B (Netplan) | 核心变更点，所有连接均指向此IP。 |
| **Neo4j Host** | 0.0.0.0 | 计算机B (Docker) | 允许远程连接，配合防火墙策略使用。 |
| **Embedding** | **Client-Side (Local)** | **计算机A (Python)** | **A机计算向量**，B机仅作存储。 |
| **Python Env** | **xiaoshuo** | 计算机A (VSCode) | 现有Python 3.11环境，用于运行逻辑脚本。 |
| **Firewall** | Allow 7474/7687/8000 | 计算机B (宝塔) | 必须在宝塔安全面板中手动放行。 |
| **Bridge Script** | Clipboard Monitor | 计算机A (Python) | 监听 >> SAVE: / >> RECALL: 实现无缝交互。 |

#### Works cited

1. AI小说写作系统框架设计.docx
2. Setting a Static IP in Ubuntu – Linux IP Address Tutorial - freeCodeCamp, accessed December 23, 2025, <https://www.freecodecamp.org/news/setting-a-static-ip-in-ubuntu-linux-ip-address-tutorial/>
3. Configuring networks - Ubuntu Server documentation, accessed December 23, 2025, <https://documentation.ubuntu.com/server/explanation/networking/configuring-networks/>
4. Setting a static IP address in Ubuntu 24.04 using `netplan` - Mendhak / Code, accessed December 23, 2025, <https://code.mendhak.com/ubuntu-2404-set-static-ip-address-using-netplan/>
5. Remote connection to Neo4j server - Stack Overflow, accessed December 23, 2025, <https://stackoverflow.com/questions/28220526/remote-connection-to-neo4j-server>
6. Configuration settings - Operations Manual - Neo4j, accessed December 23, 2025, <https://neo4j.com/docs/operations-manual/current/configuration/configuration-settings/>
7. Install - Docker Docs, accessed December 23, 2025, <https://docs.docker.com/compose/install/>
8. Running Chroma, accessed December 23, 2025, <https://cookbook.chromadb.dev/running/running-chroma/>