INSTITUTO TECNOLÓGICO SUPERIOR DE TEPOSCOLULA

REPORTE DE RESIDENCIA PROFESIONAL

CONTROL DE SISTEMAS COMPLEJOS A TRAVÉS DE MEDIOS NO LINEALES CON COMPUTACIÓN DE ALTO RENDIMIENTO

PRESENTA:
RAYMUNDO JESUS CRUZ SANTIAGO

ASESOR INTERNO:
MTIIN. MARCO ANTONIO RUIZ VICENTE

ASESOR EXTERNO:
DR. GENARO JUAREZ MARTINEZ

SAN PEDRO Y SAN PABLO TEPOSCOLULA,
OAXACA, **12 DE DICIEMBRE DE 2025**

Agradecimientos

Resumen

El presente proyecto desarrolla un simulador urbano basado en autómatas celulares para el modelado del tráfico vehicular en el municipio de Teposcolula, Oaxaca. El sistema, denominado **TeposIA**, integra una arquitectura modular que combina modelos discretos, simulación en tiempo real y procesamiento estructurado de datos urbanos, permitiendo analizar el comportamiento del tráfico desde la perspectiva de los sistemas complejos.

El desarrollo se llevó a cabo mediante una metodología híbrida compuesta por Kanban y Prototipado Evolutivo, permitiendo construir doce versiones funcionales del simulador (V1–V12), cada una con mejoras incrementales en el motor de autómata celular, la representación del mapa urbano, las probabilidades de giro, la interfaz gráfica y el manejo de múltiples carriles e intersecciones. El mapa se definió mediante un archivo JSON escalable que describe calles, nodos, flujos y direcciones cardinales.

Mediante pruebas experimentales de densidad, comportamiento en intersecciones y análisis de métricas cuantitativas, se observaron fenómenos emergentes característicos de sistemas complejos, tales como formación espontánea de colas, ondas de choque, congestión inducida y autoorganización. Los resultados obtenidos demuestran que el simulador reproduce con precisión patrones dinámicos coherentes con modelos teóricos ampliamente documentados en la literatura.

El proyecto establece una base sólida para la integración futura de modelos de inteligencia artificial, algoritmos evolutivos y técnicas de computación de alto rendimiento, con el fin de desarrollar sistemas de control inteligente del tráfico urbano en ciudades pequeñas y medianas. TeposIA constituye una herramienta versátil para análisis, docencia e investigación en el campo de la modelación computacional de sistemas complejos.

Abstract

This project presents the development of an urban traffic simulator based on cellular automata for modeling vehicular flow in the municipality of Teposcolula, Oaxaca. The system, named **TeposIA**, integrates a modular architecture combining discrete models, real-time simulation, and structured urban data processing, enabling the analysis of traffic behavior from the perspective of complex systems theory.

The simulator was developed using a hybrid methodology that combines Kanban with Evolutionary Prototyping, allowing the construction of twelve functional versions (V1–V12). Each version included incremental improvements in the cellular automaton engine, urban map representation, turning probabilities, graphical interface, lane management, and intersection behavior. The urban structure was defined through a scalable JSON format that describes streets, nodes, flows, and cardinal directions.

Experimental tests involving density analysis, intersection behavior, and quantitative metrics revealed emergent phenomena characteristic of complex systems, such as spontaneous queue formation, shockwaves, induced congestion, and self-organization. The results show that the simulator accurately reproduces dynamic patterns consistent with theoretical models documented in the literature.

This project establishes a solid foundation for future integration of artificial intelligence models, evolutionary algorithms, and high-performance computing techniques aimed at developing intelligent traffic control systems for small and medium-sized cities. TeposIA represents a versatile tool for analysis, education, and research in computational modeling of complex systems.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

GENERALIDADES DEL PROYECTO

1.1. Introducción

La comprensión y el control de los sistemas complejos representan uno de los desafíos más significativos de la ingeniería computacional contemporánea. A diferencia de los sistemas lineales, donde el comportamiento global puede deducirse de la suma de sus partes, los sistemas complejos —como el flujo vehicular urbano— exhiben dinámicas no lineales y fenómenos emergentes que escapan a los modelos matemáticos tradicionales.

El presente proyecto, titulado *Control de sistemas complejos a través de medios no lineales con computación de alto rendimiento*, aborda esta problemática mediante el diseño y desarrollo de TeposIA, un simulador de tráfico basado en autómatas celulares. Este sistema integra técnicas avanzadas de Computación de Alto Rendimiento (HPC), incluyendo paralelismo de memoria compartida y compilación Just-In-Time, para modelar con precisión la movilidad en el municipio de San Pedro y San Pablo Teposcolula, Oaxaca.

En este primer capítulo se establecen las generalidades que fundamentan el desarrollo de la residencia profesional. Se detallan los objetivos, la justificación técnica y social, así como la delimitación del alcance del proyecto, sentando las bases para una solución que combina la teoría de la complejidad con la ingeniería de software moderna.

1.2. Nombre y objetivo del proyecto

Nombre del proyecto:
Control de sistemas complejos a través de medios no lineales con computación de alto rendimiento.

Objetivo general
Desarrollar modelos de control de sistemas complejos utilizando medios no lineales y computación de alto rendimiento, aplicados al análisis y simulación de caminatas aleatorias, así como a la movilidad de individuos.

1.3. Delimitación del proyecto

1.3.1. Espacial:

El proyecto se desarrollará en el Laboratorio Internacional de Ciencias de la Computación y el Laboratorio de Vida Artificial y Robótica, ambos pertenecientes a la Escuela Superior de Cómputo (ESCOM) del Instituto Politécnico Nacional, ubicada en la Ciudad de México.

1.3.2. Temporal:

El desarrollo del proyecto está planeado para un periodo de 16 semanas, comenzando el 25 de agosto de 2025 y concluyendo el 12 de diciembre de 2025.

1.4. Objetivos

1.4.1. Objetivo general

Desarrollar modelos de control de sistemas complejos utilizando medios no lineales y computación de alto rendimiento, aplicados al análisis y simulación de caminatas aleatorias, así como a la movilidad de individuos.

1.4.2. Objetivos específicos

- Investigar y sistematizar el estado del arte sobre caminatas aleatorias y sistemas complejos.
- Diseñar algoritmos de control no lineal que modelen y simulen de forma precisa la movilidad de múltiples agentes.
- Implementar simulaciones de movilidad usando técnicas de computación de alto rendimiento en autómatas para validar los modelos desarrollados.
- Evaluar la aplicabilidad del sistema en escenarios reales de monitoreo, vigilancia y movilidad.

1.5. Justificación

El control de sistemas complejos constituye uno de los principales desafíos en la ingeniería moderna, debido a su naturaleza no lineal, alta dimensionalidad, incertidumbre y dinámicas fuertemente acopladas. Estas características dificultan el uso de técnicas clásicas de control lineal, que suponen modelos simplificados poco representativos de los sistemas reales.

Este proyecto surge del interés por encontrar soluciones eficientes al control de sistemas complejos, cada vez más presentes en áreas como la robótica, los vehículos autónomos o las redes inteligentes. Estos sistemas exhiben dinámicas no lineales y, en muchos casos, impredecibles, lo que limita la efectividad de métodos tradicionales.

Frente a ello, el uso de técnicas de control no lineal permite una mayor adaptación al comportamiento real de los sistemas. No obstante, su implementación requiere recursos computacionales intensivos, sobre todo en simulaciones detalladas o en tiempo real.

Aquí es donde la computación de alto rendimiento (HPC) se vuelve una herramienta clave, ya que permite acelerar procesos de simulación, optimización y análisis. La sinergia entre el control no lineal y la HPC permite abordar problemas de mayor complejidad, al tiempo que optimiza el desempeño y la precisión de las soluciones propuestas.

1.6. Cronograma preliminar de resultados

ACTIVIDAD	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Establecer el fondo teórico y adquirir el material necesario																
Desarrollo y prueba de algoritmos																
Presentación de resultados																
Elaboración del reporte final de residencia profesional																

1.7. Descripción detallada de las actividades

Establecer el fondo teórico y adquirir el material necesario

- Revisión bibliográfica y análisis del estado del arte.
- Selección de herramientas tecnológicas.
- Configuración del entorno de desarrollo y pruebas.

Desarrollo y prueba de algoritmos

- Diseño matemático de algoritmos de movilidad.
- Codificación e implementación de los algoritmos.
- Pruebas unitarias y simulaciones computacionales.
- Optimización mediante computación de alto rendimiento.

Presentación de resultados

- Evaluaciones experimentales en entornos controlados.
- Análisis comparativo con resultados simulados.
- Validación preliminar en escenarios reales.

Elaboración del reporte final de residencia profesional

- Redacción del documento final conforme a los lineamientos institucionales.
- Integración de resultados, gráficas y conclusiones.
- Revisión y corrección del documento.

1.8. Lugar donde se realizará el proyecto

El proyecto se desarrollará en la Escuela Superior de Cómputo (ESCOM) del Instituto Politécnico Nacional, en los siguientes laboratorios:

- Laboratorio de Vida Artificial y Robótica.
- Laboratorio Internacional de Ciencias de la Computación.

1.9. Información sobre la empresa donde se realizará el proyecto

Nombre: Instituto Politécnico Nacional – Escuela Superior de Cómputo (ESCOM)

Misión:

Formar profesionales altamente calificados en el área de ingeniería en computación, capaces de desarrollar e implementar soluciones innovadoras mediante la investigación, la ciencia y la tecnología.

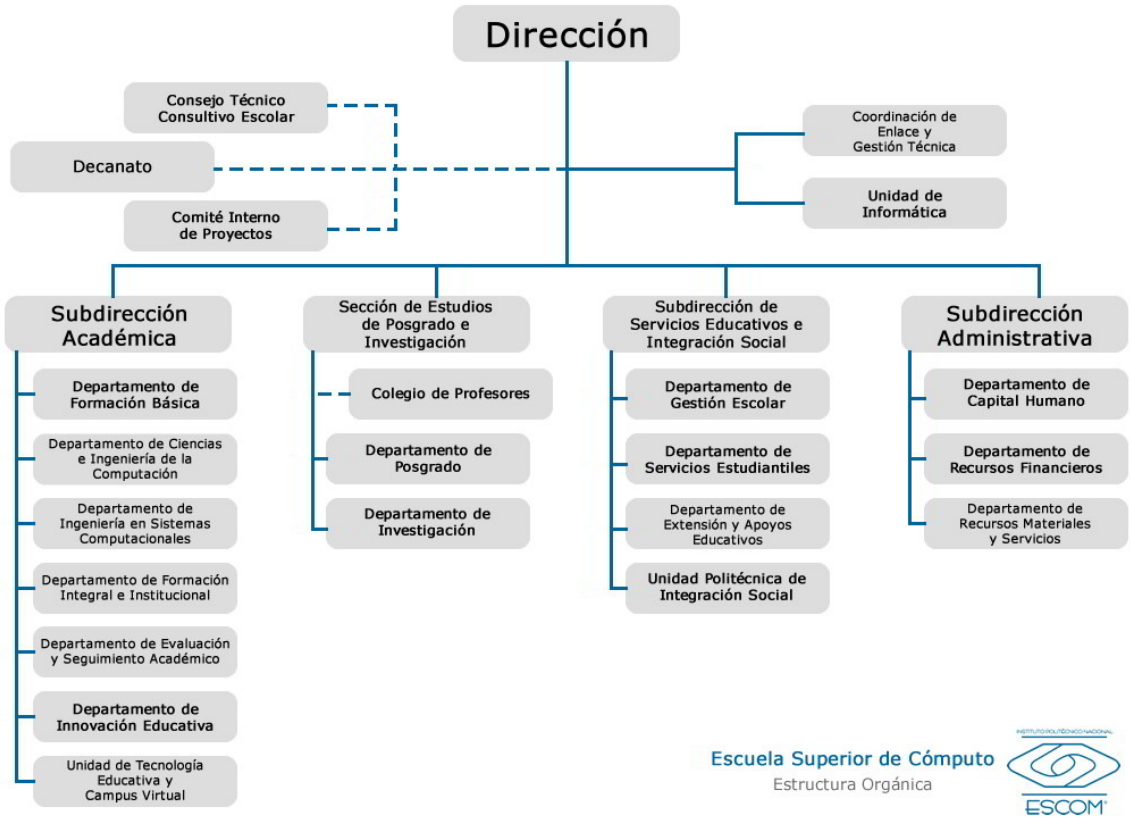
Visión:

Consolidarse como una institución de excelencia, reconocida por su impacto en el desarrollo científico y tecnológico del país, mediante la formación integral de profesionales y el impulso a la investigación aplicada.

Política del Sistema de Gestión de Organizaciones Educativas:

Nuestro compromiso es proporcionar servicios y productos educativos de calidad mediante la oferta de programas académicos, científicos y tecnológicos de educación superior pertinentes, que nos permitan contribuir con la misión y visión institucionales, con alto compromiso de responsabilidad social, procesos de transparencia, medidas de seguridad y protección de datos, así como gestión de la propiedad intelectual y la mejora continua del Sistema de Gestión para Organizaciones Educativas (SGOE) a través del cumplimiento de sus requisitos.

Organizacion:



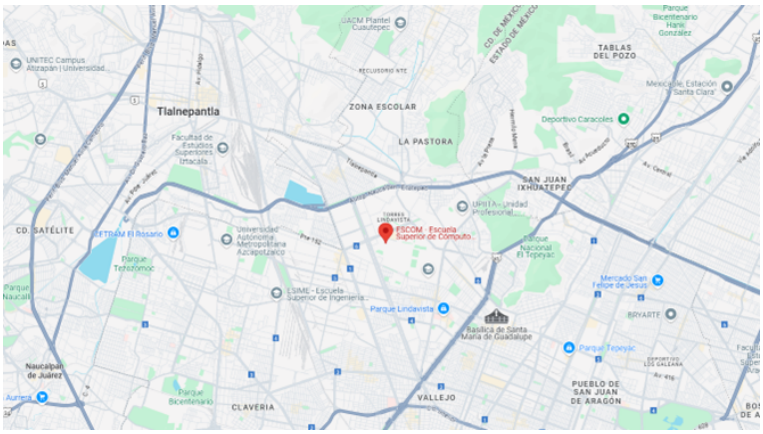
Dirección:

Av. Juan de Dios Bátiz s/n esq. Av. Miguel Othón de Mendizabal. Colonia Lindavista. Alcaldia: Gustavo A. Madero. C. P. 07738. Ciudad de México.

1.10. Macrolocalización y microlocalización

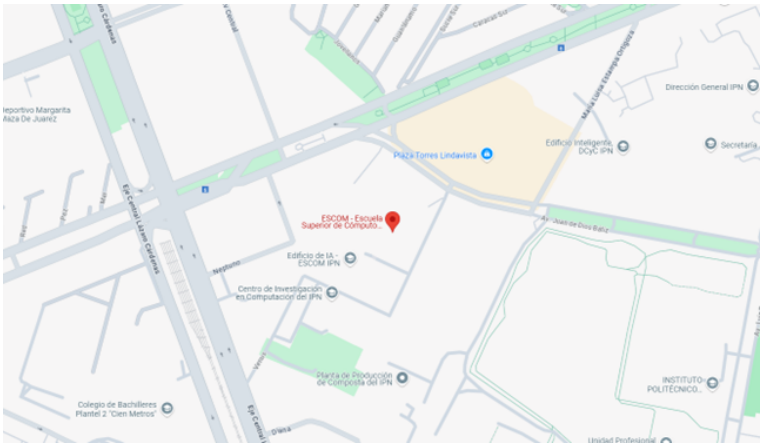
1.10.1. Macrolocalización

Ciudad de México, zona norte, alcaldía Gustavo A. Madero.



1.10.2. Microlocalización

Campus ESCOM del Instituto Politécnico Nacional.



Capítulo 2

MARCO TEÓRICO

2.1. Estado del Arte

El modelado y control del tráfico urbano es un campo de investigación activo donde convergen la ingeniería de transporte, la inteligencia artificial y la física de sistemas complejos. Para contextualizar el desarrollo de **TeposIA**, es fundamental analizar no solo las herramientas de software existentes, sino también los fundamentos teóricos que validan el uso de modelos discretos en la era de la computación de alto rendimiento.

2.1.1. Análisis de investigaciones fundamentales

El presente proyecto no surge de manera aislada, sino que se fundamenta en tres pilares de investigación académica que han evolucionado significativamente en las últimas décadas: la física del tráfico, la modelación estocástica y la computación acelerada.

La Regla 184 y la Física del Tráfico

El punto de partida teórico es el trabajo seminal de **Stephen Wolfram (2002)**, quien en su obra *A New Kind of Science* demostró que sistemas extremadamente simples pueden generar comportamientos de alta complejidad. Wolfram identificó que la *Regla 184* de los autómatas celulares elementales es isomórfica al flujo de partículas en una dimensión.

Esta investigación es crucial para el proyecto porque valida que no es necesario simular la física newtoniana de cada vehículo (fricción, aceleración mecánica) para obtener un modelo macroscópico preciso. Al abstraer el tráfico como un flujo de partículas discretas, se reduce la carga computacional en varios órdenes de magnitud comparado con simuladores continuos.

Modelos Estocásticos y Fenómenos Fantasma

Mientras que Wolfram estableció la base determinista, **Nagel y Schreckenberg (1992)** introdujeron el componente estocástico necesario para simular el error humano. Su investigación demostró que la congestión no siempre es producto de un cuello de botella físico (como un accidente), sino que puede surgir espontáneamente debido a pequeñas variaciones aleatorias en la velocidad de los conductores.

Este fenómeno, conocido como .[em]botellamiento fantasma"(*phantom traffic jam*), es uno de los comportamientos emergentes clave que este proyecto busca replicar. La inclusión de probabilidades de frenado aleatorio en el simulador permite estudiar la robustez de la red vial de Teposcolula ante perturbaciones impredecibles.

Simulación Urbana y Computación de Alto Rendimiento

En la última década (2015-2025), la investigación se ha desplazado hacia el uso de Inteligencia Artificial para el control de tráfico. Estudios recientes, como los realizados por el equipo de **CityFlow (2019)**, argumentan que los simuladores tradicionales (como SUMO) son demasiado lentos para entrenar algoritmos de Aprendizaje por Refuerzo, los cuales requieren millones de iteraciones.

Esta literatura reciente justifica la arquitectura de **HPC Híbrida** implementada en este reporte: al utilizar técnicas de vectorización (NumPy/Numba) y aceleración por GPU, se logra la velocidad necesaria para que los algoritmos de optimización converjan en tiempos viables, alineándose con el estado del arte en *Smart Cities*.

2.1.2. Resumen de literatura e impacto en el proyecto

A continuación, la Tabla 2.1 sintetiza cómo cada una de estas líneas de investigación se ha integrado específicamente en los módulos técnicos de TeposIA.

Cuadro 2.1: Vinculación de investigaciones clave con el desarrollo de TeposIA

Autor/Año	**Concepto Clave**	**Implementación en TeposIA**
Wolfram, S. (2002)	Autómatas Celulares y Clase IV (Borde del Caos).	Base del motor lógico. Se utiliza la Regla 184 para el movimiento en tramos rectos, permitiendo la vectorización del cálculo.
Nagel, K. & Schreckenberg, M. (1992)	Estocasticidad en el frenado (Modelo NaSch).	Se adaptó para las intersecciones. Los vehículos no siempre giran o avanzan óptimamente, introduciendo "ruido" realista en la simulación.
Chowdhury et al. (2000)	Física estadística del tráfico vehicular.	Fundamento para las métricas de evaluación (Densidad ρ vs Flujo J) y para identificar las fases de flujo libre y congestionado.
Zhang et al. (CityFlow, 2019)	Paralelismo en simulación para IA.	Inspiración para la arquitectura HPC. Justifica el uso de *multiprocessing* para evaluar múltiples escenarios simultáneamente.

2.1.3. Simuladores de tráfico consolidados

En la industria y la academia, existen herramientas maduras que establecen el estándar actual:

- **SUMO (Simulation of Urban MObility):** Es el estándar de código abierto más utilizado a nivel mundial. Desarrollado por el DLR (Alemania), permite simular redes multimodales masivas con un enfoque microscópico. Su principal ventaja es la extensibilidad mediante *TraCI* (Traffic Control Interface), aunque su arquitectura secuencial puede limitar el rendimiento en escenarios de entrenamiento masivo para Inteligencia Artificial.

- **PTV Vissim:** Es la herramienta comercial líder para ingeniería de tráfico. Ofrece un nivel de detalle visual y físico extremadamente alto, incluyendo el comportamiento psicofísico del conductor (modelo de Wiedemann). Sin embargo, su naturaleza de caja negra (código cerrado) y su alto costo de licencia dificultan su integración en proyectos de investigación académica que requieren modificación profunda de los algoritmos base.

- **Aimsun Next:** Destaca por su arquitectura híbrida que permite combinar modelos macroscópicos, mesoscópicos y microscópicos en una misma red. Es ampliamente utilizado para la planificación regional, aunque su curva de aprendizaje y requerimientos de hardware son elevados.

2.1.4. Nuevas tendencias: HPC e Inteligencia Artificial

En el periodo 2020-2025, ha surgido una nueva generación de simuladores diseñados específicamente para el Aprendizaje por Refuerzo (Reinforcement Learning) y la computación eficiente:

- **CityFlow:** Desarrollado para superar las limitaciones de velocidad de SUMO en contextos de aprendizaje automático. Utiliza multihilo (multi-threading) para paralelizar la actualización de vehículos, logrando ser hasta 20 veces más rápido que los simuladores tradicionales. CityFlow valida la necesidad de arquitecturas optimizadas para IA, un principio clave adoptado en este proyecto.

- **Modelos basados en Autómatas Celulares (AC):** A pesar del dominio de los modelos continuos, los AC han resurgido por su eficiencia en el manejo de *Big Data*. Investigaciones recientes (2023-2024) demuestran que los modelos discretos tipo Regla 184 son superiores para detectar transiciones de fase y fenómenos emergentes con una fracción del costo computacional de los modelos de seguimiento vehicular (car-following).

2.1.5. Análisis comparativo

La Tabla 2.2 presenta un análisis comparativo entre las principales herramientas del mercado y la solución propuesta (**TeposIA**). Se evalúan aspectos críticos como el tipo de licencia, el modelo matemático subyacente y la capacidad de integración con tecnologías de Computación de Alto Rendimiento (HPC).

Cuadro 2.2: Comparativa técnica de simuladores de tráfico urbano

Simulador	Licencia	Modelo	Lenguaje Base	Enfoque Principal	Soporte HPC/IA
SUMO	Open Source (EPL)	Micro/Meso	C++ / Python	Investigación General	Limitado (Secuencial)
PTV Vissim	Comercial	Microscópico	Propietario (COM)	Ingeniería Civil	Nulo (Caja Negra)
Aimsun	Comercial	Híbrido	C++ / Python	Planificación Regional	Limitado
CityFlow	Open Source	Microscópico	C++ / Python	Reinforcement Learning	Alto (Multihilo)
TeposIA	**Académico**	**Micro (AC)**	**Python/Numba**	**Sistemas Complejos**	**Alto (Híbrido CPU-GPU)**

2.1.6. Diferenciador del proyecto

Como se observa en la comparativa, **TeposIA** llena un nicho específico: no busca competir con la fidelidad gráfica de Vissim ni con la escala regional de Aimsun, sino ofrecer una plataforma ligera y altamente paralelizable para el estudio de la *emergencia* en sistemas complejos. A diferencia de CityFlow, que se centra en el control semafórico, TeposIA integra dinámicas estocásticas (caminatas aleatorias) y predicción neuronal en una arquitectura accesible para equipos de cómputo convencionales.

2.2. Naturaleza y fundamentos de los sistemas complejos

2.2.1. Definición general de sistema complejo

Los sistemas complejos se definen como conjuntos de múltiples componentes interactuantes cuyas relaciones dan lugar a comportamientos globales no triviales. Según Bar-Yam (1997), la complejidad surge cuando el número de componentes y la naturaleza de sus interacciones hacen imposible describir el sistema de forma reduccionista. En lugar de analizar cada parte por separado, es necesario considerar las propiedades emergentes que resultan de la interacción colectiva. Wolfram (2002) sostiene que incluso reglas locales simples, como las que gobiernan los autómatas celulares, pueden generar estructuras de sorprendente complejidad, evidenciando que la complejidad puede emerger de la simplicidad.

> "A complex system is one whose collective behavior cannot be easily inferred from the behavior of its parts" (Bar-Yam, 1997, p. 12).

2.2.2. Enfoques clásicos y contemporáneos

El estudio de los sistemas complejos ha evolucionado desde enfoques deterministas hacia modelos computacionales y autoorganizativos. Los enfoques clásicos, influenciados por la física newtoniana, asumían sistemas lineales y predecibles. Sin embargo, los enfoques contemporáneos, como los de Wolfram (2002) y Mainzer & Chua (2011), reconocen la existencia de comportamientos caóticos, autoorganización y patrones emergentes que surgen de interacciones no lineales.

Bar-Yam (1997) introduce un marco interdisciplinario basado en física estadística, biología y teoría de redes para analizar sistemas complejos dinámicos, mientras que Adamatzky (2010) y Griffeath & Moore (1996) utilizan autómatas celulares como herramientas de simulación para explorar fenómenos complejos en contextos computacionales y naturales.

2.2.3. Características esenciales

Las propiedades fundamentales que caracterizan a los sistemas complejos incluyen:

- **Emergencia:** Propiedad por la cual patrones globales surgen de reglas locales simples sin control central (Wolfram, 2002).
- **Adaptación:** Capacidad del sistema para modificar su comportamiento frente a cambios del entorno o perturbaciones externas (Bar-Yam, 1997).
- **Robustez:** Resistencia del sistema frente a fallos o pérdida de componentes, manteniendo su funcionalidad global (Mainzer & Chua, 2011).
- **No linealidad:** Relación no proporcional entre causas y efectos, que genera fenómenos impredecibles o caóticos (Bar-Yam, 1997).
- **Interconectividad:** Las dependencias entre elementos determinan la dinámica global del sistema y la transmisión de información (Adamatzky et al., 2008).

En palabras de Mainzer y Chua (2011), la complejidad no reside en las partes, sino en las *relaciones* que las conectan: "The key to understanding complexity lies not in the components themselves but in their interactions" (p. 48).

2.2.4. Representación formal del sistema complejo

Formalmente, un sistema complejo puede representarse como:

$$S = \{e_i, f(e_i, e_j), P\}$$

donde e_i representa los elementos o agentes del sistema, $f(e_i, e_j)$ las funciones de interacción entre ellos, y P el conjunto de parámetros o propiedades emergentes globales (Bar-Yam, 1997). Este enfoque es compatible con los modelos de autómatas celulares propuestos por Wolfram (2002) y Adamatzky (2010), donde las interacciones locales determinan el estado global del sistema.

2.2.5. Ejemplos paradigmáticos

Entre los ejemplos más representativos de sistemas complejos se encuentran los ecosistemas, las redes neuronales, el tráfico vehicular y los sistemas sociales. Wolfram (2002) demuestra que los autómatas celulares de tipo Regla 184 pueden modelar flujos vehiculares, mientras que Bar-Yam (1997) los utiliza para analizar la dinámica de poblaciones. Adamatzky (2010) extiende estos modelos al ámbito de la inteligencia artificial y la morfogénesis. En las ciencias sociales, Mainzer y Chua (2011) destacan cómo las interacciones humanas generan estructuras económicas y culturales emergentes.

2.2.6. Complejidad computacional vs. complejidad dinámica

La **complejidad computacional** se refiere al esfuerzo algorítmico requerido para describir o simular un sistema, como se estudia en los autómatas celulares de Wolfram (2002) o en los trabajos de Griffeath y Moore (1996) sobre reglas locales y clases de comportamiento. En cambio, la **complejidad dinámica** describe la evolución temporal de un sistema bajo interacciones no lineales, aspecto central en los estudios de Bar-Yam (1997) y Mainzer & Chua (2011). Ambos enfoques se complementan: mientras el primero evalúa la dificultad de cálculo, el segundo examina la evolución de los patrones emergentes.

2.3. Dinámica no lineal y autoorganización

2.3.1. Concepto de no linealidad en sistemas dinámicos

Un sistema dinámico se considera no lineal cuando su comportamiento no puede expresarse como una combinación lineal de sus variables o estados. En otras palabras, pequeñas variaciones en las condiciones iniciales pueden provocar grandes diferencias en la evolución del sistema. Según Bar-Yam (1997), la no linealidad es la fuente fundamental de la complejidad, ya que impide predecir el comportamiento global mediante métodos reduccionistas.

Mainzer y Chua (2011) señalan que los sistemas no lineales pueden exhibir comportamientos caóticos o autoorganizativos, dependiendo de la estructura de sus ecuaciones diferenciales. En contraste con los sistemas lineales, donde las soluciones son proporcionales a las entradas, los sistemas no lineales presentan interacciones múltiples, realimentaciones internas y comportamientos emergentes.

> "Nonlinear dynamics reveals that determinism does not imply predictability; small perturbations may lead to entirely new structures" (Mainzer & Chua, 2011, p. 63).

2.3.2. Caos determinista y sensibilidad a condiciones iniciales

El concepto de caos determinista fue introducido por Edward Lorenz (1963) al estudiar la convección atmosférica mediante ecuaciones diferenciales simples. Lorenz descubrió que incluso sistemas gobernados por leyes deterministas pueden presentar comportamientos impredecibles debido a su extrema sensibilidad a las condiciones iniciales. Esta propiedad, conocida como el *efecto mariposa*, implica que una mínima variación en el estado inicial puede conducir a trayectorias divergentes en el tiempo.

Wolfram (2002) argumenta que los autómatas celulares también exhiben caos determinista, demostrando que incluso reglas discretas simples pueden generar comportamientos caóticos sin necesidad de ecuaciones continuas. Este tipo de comportamiento es característico de los sistemas de clase III y IV según su clasificación.

2.3.3. Atractores, bifurcaciones y estabilidad

En la teoría de sistemas dinámicos, los **atractores** representan los estados hacia los cuales un sistema tiende a evolucionar con el tiempo. Lorenz (1963) identificó el célebre *atractor de Lorenz*, una estructura fractal que ilustra el equilibrio entre orden y caos.

Las **bifurcaciones** son transiciones críticas donde pequeñas variaciones en parámetros producen cambios cualitativos en el comportamiento del sistema, como pasar de un ciclo estable a un régimen caótico. Mainzer y Chua (2012) explican que la estabilidad local de un sistema puede perderse cuando se rompe la simetría, dando origen a nuevas estructuras autoorganizadas. Chua (1992) demostró este fenómeno en su circuito no lineal, conocido como el *circuito de Chua*, el cual constituye uno de los primeros ejemplos físicos de caos electrónico.

2.3.4. Fenómenos autoorganizativos en la naturaleza y la ingeniería

La autoorganización se refiere a la capacidad de un sistema para generar estructuras ordenadas sin control externo, emergiendo a partir de interacciones locales. Prigogine y Stengers (1984) introducen el concepto de *estructuras disipativas*, donde sistemas alejados del equilibrio termodinámico pueden estabilizar patrones espacio-temporales. Ejemplos naturales incluyen la formación de dunas, patrones biológicos y la organización de colonias bacterianas.

Mainzer y Chua (2011) proponen que la autoorganización es una consecuencia de la realimentación no lineal y la ruptura de simetrías, un mecanismo universal que también se observa en ingeniería, como en redes neuronales artificiales o en osciladores sincronizados. Bar-Yam (1997) sostiene que estos fenómenos pueden modelarse computacionalmente como sistemas adaptativos complejos.

2.3.5. Modelos clásicos de sistemas no lineales

Entre los modelos más representativos de la dinámica no lineal se encuentran:

- **Sistema de Lorenz** (Lorenz, 1963): describe la convección térmica en fluidos; muestra caos determinista y atractores extraños.
- **Circuito de Chua** (Chua, 1992): modelo electrónico no lineal que genera oscilaciones caóticas mediante realimentación negativa.
- **Modelos de reacción-difusión** (Turing, 1952; Mainzer & Chua, 2011): explican la formación de patrones químicos y biológicos por medio de interacciones locales y difusión espacial.

Estos modelos demuestran que la complejidad puede emerger tanto en sistemas naturales como artificiales, confirmando la hipótesis de Wolfram (2002) sobre la universalidad computacional de las reglas locales.

2.3.6. El papel de la realimentación y la simetría rota (Mainzer & Chua, 2012)

Mainzer y Chua (2012) sostienen que la realimentación (*feedback*) constituye el núcleo de la autoorganización. La simetría rota aparece cuando el sistema, inicialmente homogéneo, evoluciona hacia un estado diferenciado como resultado de inestabilidades internas. Este principio explica la aparición de estructuras en diversos dominios: desde cristales hasta circuitos caóticos.

> "Broken symmetries are not signs of disorder but the seeds of organized complexity" (Mainzer & Chua, 2012, p. 97).

En la ingeniería moderna, los sistemas de control no lineal aprovechan estos principios para diseñar redes estables y adaptativas, mientras que en física y biología se observan patrones similares en oscilaciones químicas o dinámicas neuronales.

2.3.7. Ejemplos de autoorganización

Diversos fenómenos naturales y tecnológicos ilustran la autoorganización:

- **Ondas de tráfico:** modeladas mediante autómatas celulares tipo Regla 184 (Wolfram, 2002), representan la emergencia espontánea de congestiones y su propagación colectiva.
- **Patrón de convección de Bénard:** ejemplo clásico de estructura disipativa donde el flujo térmico induce células hexagonales autoorganizadas (Prigogine & Stengers, 1984).
- **Sincronización neuronal:** las oscilaciones de fase en redes neuronales exhiben autoorganización funcional mediante acoplamiento dinámico (Mainzer & Chua, 2011).

Estos ejemplos muestran que la autoorganización constituye un principio unificador que conecta la física, la biología y la ingeniería bajo el marco de la complejidad no lineal.

2.4. Control de sistemas no lineales

2.4.1. Fundamentos del control moderno

El control moderno se basa en el análisis matemático de sistemas dinámicos utilizando ecuaciones diferenciales y técnicas de retroalimentación para regular su comportamiento. A diferencia del control clásico, que se apoya en aproximaciones lineales y en el dominio de la frecuencia, el control moderno se formula en el dominio del estado, permitiendo el diseño de controladores que operan sobre sistemas multivariables y no lineales (Slotine & Li, 1991).

Mainzer y Chua (2012) destacan que la teoría del control es una extensión natural de la dinámica no lineal: los principios de realimentación, estabilidad y autoorganización se encuentran profundamente relacionados. En los sistemas complejos, la retroalimentación constituye el mecanismo que regula la evolución y estabiliza el comportamiento global ante perturbaciones locales.

> "Feedback transforms instability into structure; it is the bridge between chaos and control" (Mainzer & Chua, 2012, p. 154).

2.4.2. Control lineal vs. control no lineal

El control lineal asume que las relaciones entre las variables de entrada y salida del sistema pueden aproximarse mediante modelos lineales. Esta aproximación es válida únicamente en torno a puntos de equilibrio y bajo perturbaciones pequeñas. En cambio, el control no lineal aborda sistemas cuya dinámica presenta dependencias cuadráticas, cúbicas o saturaciones, lo que impide el uso de métodos lineales convencionales (Khalil, 2002).

Bar-Yam (1997) argumenta que los sistemas complejos del mundo real —como ecosistemas, redes de tráfico o sistemas neuronales— requieren estrategias de control no lineal debido a su interdependencia y sensibilidad a condiciones iniciales. De modo similar, Chua (1992) demuestra que la presencia de no linealidades locales puede inducir comportamientos globales caóticos que sólo pueden regularse mediante técnicas adaptativas o de realimentación no lineal.

2.4.3. Técnicas principales

Las estrategias más relevantes para el control de sistemas no lineales incluyen:

- **Linealización por realimentación:** transforma el sistema no lineal en uno lineal equivalente mediante una ley de control basada en la cancelación de términos no lineales. Es efectiva cuando el modelo del sistema es conocido (Isidori, 1995).
- **Control deslizante (Sliding Mode Control):** técnica robusta que obliga al sistema a evolucionar sobre una superficie de deslizamiento en el espacio de estados, insensible a incertidumbres del modelo (Utkin, 1992).
- **Control adaptativo:** ajusta los parámetros del controlador en tiempo real según la respuesta del sistema, lo que permite manejar dinámicas cambiantes (Åström & Wittenmark, 2013).
- **Control predictivo (MPC):** optimiza la trayectoria futura del sistema considerando restricciones y objetivos múltiples; resulta útil en procesos industriales y sistemas distribuidos (Camacho & Bordons, 2007).

Cada una de estas técnicas refleja un principio de autoorganización controlada, donde la retroalimentación y la predicción actúan como mecanismos de estabilidad en entornos complejos (Mainzer & Chua, 2011).

2.4.4. Control distribuido y descentralizado

El control distribuido surge como respuesta a la creciente escala y complejidad de los sistemas modernos. En lugar de emplear un controlador central, múltiples controladores locales coordinan su comportamiento a través de redes de comunicación. Bar-Yam (1997) describe este paradigma como un ejemplo de *control emergente*, en el cual las decisiones locales producen coherencia global sin un centro jerárquico.

Ejemplos incluyen las redes eléctricas inteligentes (*smart grids*), el tráfico vehicular autónomo y los enjambres robóticos. Según Wolfram (2002), este tipo de control descentralizado se asemeja a los autómatas celulares: reglas locales simples generan dinámicas coordinadas a nivel global.

> "Distributed control represents the engineering analog of self-organization in complex systems" (Bar-Yam, 1997, p. 314).

2.4.5. Relación entre control no lineal y medios complejos

La teoría de medios complejos analiza sistemas formados por elementos interconectados cuyas interacciones producen propiedades colectivas no triviales. Mainzer y Chua (2012) sostienen que el control no lineal permite gestionar la complejidad de dichos sistemas mediante mecanismos de realimentación jerárquica, donde el comportamiento emergente se mantiene dentro de límites deseados.

En la perspectiva de Wolfram (2002), los controladores pueden verse como "reglas adicionales" que alteran el comportamiento de un autómata celular, modificando su clase de evolución (del caos hacia el orden). Este enfoque es especialmente relevante para la regulación de redes dinámicas, tráfico o poblaciones artificiales.

2.4.6. Aplicaciones del control en sistemas emergentes

El control no lineal tiene aplicaciones fundamentales en sistemas emergentes y caóticos:

- **Robótica:** Los controladores adaptativos permiten el movimiento preciso de robots manipuladores con dinámicas altamente no lineales (Slotine & Li, 1991).
- **Tráfico vehicular:** Modelos basados en la Regla 184 y control distribuido de flujo permiten regular la densidad de vehículos mediante nodos inteligentes (Wolfram, 2002; Bar-Yam, 1997).
- **Dinámica caótica:** Técnicas como el control de Ott-Grebogi-Yorke (OGY) estabilizan órbitas inestables en sistemas caóticos como el circuito de Chua o el atractor de Lorenz (Ott, Grebogi, & Yorke, 1990).

Estas aplicaciones evidencian que el control no lineal no sólo busca eliminar el caos, sino aprovecharlo como fuente de adaptabilidad y creatividad en sistemas físicos y computacionales (Mainzer & Chua, 2012).

2.5. Computación natural y paradigma de Wolfram

2.5.1. Origen de la computación natural

La computación natural surge como un paradigma interdisciplinario que busca inspirarse en los procesos de la naturaleza para desarrollar nuevos métodos de procesamiento de información. Según Adamatzky (2010), este enfoque se basa en la

idea de que los fenómenos físicos, biológicos y químicos pueden interpretarse como sistemas de cómputo distribuidos.

Mainzer y Chua (2011) argumentan que la computación natural representa una extensión del pensamiento sistémico, donde el cálculo deja de ser un proceso exclusivamente digital o simbólico para convertirse en una manifestación de la autoorganización y la dinámica no lineal. En este sentido, los sistemas naturales no sólo "se comportan" sino que "procesan información" a través de sus interacciones.

> "Nature itself computes; every process of self-organization is a computation of physical laws." (Mainzer & Chua, 2011, p. 113)

2.5.2. Stephen Wolfram y *A New Kind of Science*

En 2002, Stephen Wolfram publicó *A New Kind of Science*, obra en la que propone un nuevo paradigma científico basado en la exploración computacional de sistemas simples. Wolfram sostiene que las reglas locales y discretas —como las que gobiernan los autómatas celulares— son suficientes para generar toda la diversidad de comportamientos observables en la naturaleza (Wolfram, 2002).

Este enfoque desafía la tradición analítica de la ciencia matemática clásica, proponiendo que los modelos computacionales son más adecuados para describir fenómenos complejos que las ecuaciones diferenciales tradicionales. Bar-Yam (1997) coincide en que la simulación computacional constituye una herramienta esencial para comprender sistemas con múltiples niveles de interacción y emergencia.

2.5.3. El universo como autómata celular

Wolfram (2002) plantea la hipótesis de que el universo puede entenderse como un gran autómata celular, donde cada célula representa una unidad fundamental del espacio-tiempo que evoluciona según reglas locales deterministas. Este modelo se sustenta en la idea de que la complejidad del cosmos no requiere un diseño previo, sino que emerge de interacciones simples repetidas a lo largo del tiempo.

Mainzer y Chua (2011) amplían este argumento al afirmar que los autómatas celulares constituyen una forma de *computación física*, en la que las leyes naturales actúan como programas que ejecutan la evolución del universo. De esta manera, la realidad se concibe como un proceso computacional autoorganizado.

> "If the universe is a computation, then every particle, every interaction, is a step of that program." (Wolfram, 2002, p. 470)

2.5.4. Regla 110 y Regla 184: modelos de dinámica conservativa y universalidad

Entre los autómatas unidimensionales más relevantes descritos por Wolfram se encuentran la Regla 110 y la Regla 184. La **Regla 110** es especialmente significativa por su *universalidad computacional*; Cook (2004) demostró que puede simular cualquier máquina de Turing, lo que la convierte en un modelo fundamental de computación emergente. La **Regla 184**, en cambio, representa un sistema conservativo utilizado para modelar el tráfico vehicular y los flujos unidimensionales de partículas (Wolfram, 2002; Adamatzky et al., 2008).

Ambas reglas ilustran cómo los comportamientos complejos —orden, caos y patrones recurrentes— pueden surgir a partir de estructuras locales mínimas, confirmando que la complejidad no requiere aleatoriedad externa sino sólo interacciones deterministas adecuadas.

2.5.5. Clasificación de Wolfram (Clases I–IV)

Wolfram (1984, 2002) propuso una taxonomía de los autómatas celulares basada en su comportamiento dinámico:

- **Clase I:** los sistemas evolucionan hacia un estado homogéneo estable (orden total).
- **Clase II:** aparecen patrones periódicos o estáticos (estructuras regulares).
- **Clase III:** exhiben comportamiento caótico y pseudoaleatorio.
- **Clase IV:** muestran una combinación de orden y caos, generando estructuras complejas y propagación de información (vida artificial, computación emergente).

La Regla 110 pertenece a la Clase IV, lo que la sitúa en el umbral entre el orden y el caos, una región que Wolfram denomina *edge of chaos*, considerada el punto óptimo para la aparición de inteligencia y autoorganización.

2.5.6. Comparación entre modelos continuos y discretos

En la física tradicional, los sistemas se describen mediante ecuaciones diferenciales continuas. Sin embargo, Wolfram (2002) y Mainzer & Chua (2011) señalan que los modelos discretos, como los autómatas celulares, ofrecen ventajas conceptuales: son simples, universales y evitan los problemas de cálculo infinitesimal en procesos caóticos.

Mientras los modelos continuos buscan la precisión matemática, los modelos discretos enfatizan la evolución paso a paso, donde cada actualización computa una "versión digital" de la realidad. Esta perspectiva permite estudiar fenómenos que antes eran inabordables, como la emergencia de patrones o la evolución de sistemas fuera del equilibrio.

2.5.7. Computación basada en reglas locales

El principio fundamental de la computación de Wolfram es la **regla local**: cada célula o componente del sistema actualiza su estado sólo a partir de la información de sus vecinos inmediatos. Este paradigma refleja una forma descentralizada de procesamiento, similar al control distribuido en sistemas complejos (Bar-Yam, 1997).

Adamatzky (2008) demuestra que mediante la combinación de reglas locales pueden construirse redes de procesamiento equivalentes a circuitos lógicos, autómatas universales o simulaciones biológicas. En consecuencia, la computación basada en reglas locales constituye un puente entre la física, la biología y la ingeniería.

2.5.8. Filosofía computacional: del determinismo al comportamiento emergente

La filosofía de Wolfram representa una ruptura con el paradigma determinista clásico. Aunque sus autómatas son sistemas completamente deterministas, sus resultados son impredecibles y muestran comportamientos emergentes que sólo pueden comprenderse mediante la observación computacional.

Mainzer y Chua (2012) interpretan este cambio como una transición del *determinismo predictivo* al *determinismo generativo*, donde las leyes simples generan complejidad sin requerir azar. Esto redefine el concepto de causalidad: el universo no se calcula mediante ecuaciones, sino que *se ejecuta* como un programa.

> "We must abandon the idea that predictability defines science; computation shows that unpredictability is an intrinsic property of deterministic systems." (Wolfram, 2002, p. 1020)

Este enfoque establece un nuevo paradigma para la comprensión de los sistemas complejos, en el que la computación y la naturaleza convergen en una misma ontología: la del universo como proceso algorítmico.

2.6. Autómatas celulares: teoría y aplicaciones

2.6.1. Definición formal y fundamentos matemáticos

Los autómatas celulares (AC) constituyen uno de los modelos fundamentales de la computación natural. Se definen como sistemas dinámicos discretos formados por una retícula de celdas que evolucionan en el tiempo según una regla local de transición dependiente del estado de sus vecinas. Formalmente, un autómata celular puede representarse como una 7-tupla:

$$A = \langle L, S, N, \delta, t_0, f, B \rangle$$

donde:

- L es el conjunto de posiciones (celdas) dispuestas en una red discreta.
- S es el conjunto finito de estados posibles para cada celda.
- N define la vecindad (von Neumann o Moore, entre otras).
- $\delta : S^{|N|} \to S$ es la función local de transición.
- t_0 representa el estado inicial.
- f es la función de evolución temporal.
- B son las condiciones de frontera.

Según Wolfram (2002), los autómatas celulares permiten explorar el principio de que reglas simples pueden generar una enorme diversidad de comportamientos complejos, lo que los convierte en herramientas esenciales para estudiar la emergencia, la autoorganización y la complejidad computacional.

2.6.2. Vecindades de von Neumann y Moore

La **vecindad de von Neumann** considera las cuatro celdas ortogonales (norte, sur, este y oeste) alrededor de una celda central. En cambio, la **vecindad de Moore** incluye además las diagonales, totalizando ocho vecinos.

La elección de vecindad determina la topología de las interacciones y, por tanto, el comportamiento global del sistema. Según Adamatzky (2010), la vecindad de Moore favorece patrones isotrópicos y difusivos, mientras que la de von Neumann resalta la anisotropía y las dinámicas direccionales, características útiles para el modelado de flujos, propagación o tráfico vehicular.

2.6.3. Tipos de autómatas

La taxonomía de los AC depende de su estructura y de la naturaleza de su función de transición:

- **Elementales:** unidimensionales, con dos estados y vecindad de tres celdas. Ejemplo paradigmático: la Regla 110, universal en el sentido de Turing (Cook, 2004).
- **Bidimensionales:** como el *Game of Life* de Conway, que exhibe comportamientos autoorganizativos y estructuras móviles (Adamatzky, 2010).
- **Probabilísticos:** incorporan ruido o probabilidad de transición, permitiendo modelar fenómenos estocásticos como epidemias o incendios (Lawniczak & Kapral, 1993).
- **Reversibles:** conservan información y energía; utilizados para representar procesos físicos sin pérdida (Morita, 2013).

Los trabajos de Griffeath y Moore (1996) muestran que los autómatas elementales y bidimensionales comparten propiedades de universalidad computacional y transiciones de fase análogas a las observadas en sistemas físicos no lineales.

2.6.4. Propiedades fundamentales

Los AC presentan tres propiedades clave que los diferencian de otros modelos de simulación:

- **Homogeneidad:** todas las celdas siguen la misma regla de evolución, reflejando la invariancia espacial del sistema.
- **Localismo:** cada celda interactúa sólo con sus vecinos inmediatos, lo que permite una implementación distribuida y paralela.
- **Sincronía:** la actualización ocurre simultáneamente en todas las celdas, generando una dinámica discreta por pasos temporales.

Estas propiedades garantizan que el comportamiento global emerja de interacciones locales, un principio que Wolfram (1984) considera la base de la complejidad natural.

2.6.5. Aplicaciones multidisciplinares

Los autómatas celulares se han consolidado como herramientas universales para la modelación de procesos naturales y artificiales. Algunas de sus aplicaciones más relevantes incluyen:

- **Reacción–difusión y procesos químicos:** Adamatzky (2005) desarrolló modelos de reacción–difusión basados en AC para estudiar la propagación de ondas químicas y la formación de patrones de Turing.
- **Propagación de incendios y epidemias:** los AC probabilísticos permiten simular la extensión de incendios forestales y contagios epidémicos mediante reglas locales de propagación (Lawniczak & Kapral, 1993).
- **Modelado del tráfico vehicular:** la Regla 184, el modelo BML (Biham–Middleton–Levine) y el NaSch (Nagel–Schreckenberg) describen el flujo de vehículos en calles o autopistas, representando un paradigma de autoorganización y congestión (Nagel & Schreckenberg, 1992; Wolfram, 2002; Martínez et al., 2013).
- **Robótica y control distribuido:** los AC se aplican al control cooperativo de enjambres de robots y sistemas de navegación autónoma, aprovechando la comunicación local entre agentes (Adamatzky et al., 2008).
- **Dinámica de fluidos y morfogénesis:** los modelos de tipo *lattice gas* y *lattice Boltzmann* derivan directamente de AC y permiten simular la dinámica de fluidos, crecimiento biológico y formación de estructuras (Chopard & Droz, 1998).

Estas aplicaciones demuestran la versatilidad de los AC como lenguaje común entre la física, la biología, la ingeniería y las ciencias computacionales.

2.6.6. Investigaciones clave

Las contribuciones contemporáneas más influyentes en el estudio de los AC provienen de diversos autores y laboratorios internacionales:

- **Andrew Adamatzky** (Universidad del West of England) ha desarrollado la teoría de los AC en medios no lineales, computación no convencional y autómatas de reacción–difusión (Adamatzky, 2005, 2010).
- **Genaro J. Martínez** (UNAM – UASLP) ha contribuido al análisis formal de autómatas elementales y sus equivalencias lógicas, incluyendo la Regla 110 y los mecanismos de colisión de partículas (Martínez et al., 2013).
- **David Griffeath y Christopher Moore** (Universidad de Wisconsin y Santa Fe Institute) profundizaron en la universalidad y dinámica estadística de AC (Griffeath & Moore, 1996).
- **Kenichi Morita** exploró la reversibilidad, conservación y computación reversible en AC (Morita, 2013).
- **Anna Lawniczak** analizó modelos probabilísticos de tráfico y difusión con AC en contextos urbanos y de transporte (Lawniczak & Kapral, 1993).

Estos trabajos consolidan la visión de los autómatas como un marco matemático unificador para el estudio de sistemas dinámicos discretos y complejos.

2.6.7. Computación en medios no lineales (Adamatzky, 2005)

Adamatzky (2005) propone el concepto de **computación en medios no lineales**, en el que los procesos de cómputo se realizan mediante la propagación de ondas, excitaciones o reacciones químicas. En este paradigma, la información no se transmite de manera digital, sino como perturbaciones físicas que interactúan y se transforman, reproduciendo la dinámica de un autómata celular continuo.

Este enfoque vincula la física no lineal con la teoría de la computación: las colisiones entre frentes de onda pueden implementarse como operaciones lógicas o sistemas de control distribuido. Mainzer y Chua (2012) interpretan esta perspectiva como una manifestación del principio de *actividad local*, donde la complejidad surge de interacciones microscópicas en medios físicos reales.

En consecuencia, los AC no sólo son modelos teóricos, sino que constituyen una base experimental para el desarrollo de *hardware no lineal*, biocomputación y sistemas de control autónomo inspirados en la naturaleza.

2.7. El Juego de la Vida y la emergencia computacional

2.7.1. John Conway y la génesis del modelo (Gardner, 1970)

El *Juego de la Vida* (*Game of Life*) fue concebido en 1970 por el matemático británico John Horton Conway y divulgado por Martin Gardner en la revista *Scientific American*. Este modelo introdujo un paradigma completamente nuevo en la comprensión de los sistemas dinámicos discretos: una simulación simple capaz de generar estructuras de sorprendente complejidad sin intervención externa (Gardner, 1970).

Conway desarrolló el modelo como un ejemplo de cómo la autoorganización y la evolución compleja pueden surgir de reglas locales y deterministas. Según Wolfram (2002), el Juego de la Vida representa uno de los descubrimientos experimentales más importantes en el estudio de la computación natural, ya que demuestra la existencia de comportamientos emergentes no predecibles a partir de las reglas iniciales.

> "Conway's Life proved that simplicity can give rise to infinite complexity —a computational universe born from four local rules." (Wolfram, 2002, p. 872)

2.7.2. Definición formal y reglas de transición

El Juego de la Vida es un autómata celular bidimensional definido sobre una retícula cuadrada infinita, donde cada célula puede encontrarse en uno de dos estados: viva (1) o muerta (0). Su evolución temporal se determina por la vecindad de Moore (ocho celdas adyacentes) y cuatro reglas locales simples (Conway, 1970):

1. Una célula viva con dos o tres vecinas vivas sobrevive.

2. Una célula muerta con exactamente tres vecinas vivas nace (reproducción).

3. Las demás células mueren por soledad o sobrepoblación.

4. Todas las celdas se actualizan simultáneamente en cada paso de tiempo.

Formalmente, si $C_t(x, y)$ representa el estado de la célula (x, y) en el tiempo t, entonces la regla puede expresarse como:

$$C_{t+1}(x,y) = \begin{cases} 1, & \text{si } C_t(x,y) = 1 \text{ y } n_t(x,y) \in \{2,3\}, \\ 1, & \text{si } C_t(x,y) = 0 \text{ y } n_t(x,y) = 3, \\ 0, & \text{en otro caso,} \end{cases}$$

donde $n_t(x, y)$ es el número de vecinos vivos en la vecindad de Moore.

2.7.3. Fenómenos emergentes

De la combinación de estas reglas surgen patrones dinámicos conocidos como *gliders*, *cañones*, *osciladores* y estructuras denominadas *metapatrones* o *macroautómatas*. Adamatzky (2010) describe estos patrones como manifestaciones de computación emergente, donde las colisiones e interacciones locales producen información y orden a gran escala.

- **Gliders:** estructuras móviles que se desplazan diagonalmente por el espacio; representan la unidad básica de transporte de información.
- **Cañones (Gosper Gun):** generan una secuencia infinita de gliders, demostrando la posibilidad de un crecimiento ilimitado.
- **Osciladores:** patrones que repiten su configuración en ciclos temporales fijos.
- **Metapatrones:** combinaciones jerárquicas de estructuras que forman sistemas autoorganizados de segundo orden (Adamatzky, 2010).

Estos fenómenos confirman que la emergencia no requiere aleatoriedad, sino que puede originarse en la interacción determinista de componentes simples (Wolfram, 2002).

2.7.4. Turing-completitud y computación universal

Uno de los descubrimientos más notables del Juego de la Vida es su **Turing-completitud**. Berlekamp, Conway y Guy (1982) demostraron que, mediante una combinación adecuada de gliders, puertas lógicas y cañones, es posible implementar una máquina de Turing universal dentro del autómata.

Adamatzky (2010) amplía esta noción bajo el concepto de *computación en colisión*, donde las interacciones entre patrones móviles equivalen a operaciones lógicas. Este hallazgo convierte al Juego de la Vida en un modelo de computación emergente, donde el procesamiento de información surge espontáneamente de la dinámica del sistema.

> "In Life, logic and computation are embodied in the motion and collision of localized structures." (Adamatzky, 2010, p. 26)

2.7.5. Simulación de patrones y caos controlado

El Juego de la Vida se caracteriza por un equilibrio entre orden y caos. Los patrones iniciales pueden evolucionar hacia estabilidad, periodicidad o comportamiento caótico, dependiendo de su configuración. Wolfram (2002) clasifica al Juego de la Vida dentro de la **Clase IV**, donde los sistemas exhiben una mezcla de orden estructurado y desorden aparente, lo que los convierte en el entorno ideal para el estudio de la emergencia.

En el contexto del control no lineal, este tipo de sistemas pueden utilizarse como laboratorios digitales para analizar la estabilidad, la entropía y la propagación de información. Adamatzky y Martínez (2013) demostraron que mediante la manipulación selectiva de gliders es posible inducir comportamientos controlados, abriendo el campo de la *computación programable en autómatas*.

2.7.6. Aplicaciones contemporáneas en IA y física computacional

El Juego de la Vida ha trascendido el ámbito teórico para convertirse en un modelo de referencia en la inteligencia artificial y la física computacional. En IA, se utiliza para estudiar la emergencia de comportamiento colectivo, optimización distribuida y aprendizaje evolutivo. En física, se aplica al análisis de sistemas fuera del equilibrio, autómatas de reacción–difusión y dinámica de redes neuronales artificiales (Adamatzky, 2010; Mainzer & Chua, 2011).

En contextos HPC (High Performance Computing), la simulación masiva de autómatas tipo Life permite investigar la relación entre paralelismo, entropía y eficiencia computacional. El paralelismo inherente a la actualización sincrónica de celdas hace del modelo un entorno ideal para arquitecturas distribuidas y aceleradores GPU.

2.7.7. Rol del Juego de la Vida en la presente investigación

En la presente investigación, el Juego de la Vida se adopta como un entorno de validación para el estudio del **control en sistemas no lineales y emergentes**, debido a su equilibrio entre orden y caos y su capacidad para representar fenómenos complejos a partir de reglas simples.

El modelo se emplea como base para evaluar:

- La propagación de información en sistemas autoorganizados.
- El comportamiento de la entropía y la densidad bajo condiciones de control.
- La escalabilidad de simulaciones HPC en mallas de gran tamaño.

El Juego de la Vida constituye así un **marco experimental computacional** que permite explorar los principios de realimentación, autoorganización y control distribuido en sistemas complejos, sirviendo como analogía directa con los fenómenos de tráfico, redes neuronales y control adaptativo desarrollados en capítulos posteriores.

2.8. Modelos de tráfico y la Regla 184

2.8.1. Modelo Matemático: La Regla 184

La **Regla 184**, propuesta originalmente por Wolfram (1983, 2002), es un autómata celular elemental unidimensional reconocido en la literatura científica como el modelo determinista más simple capaz de reproducir las características esenciales del flujo vehicular.

El modelo representa una vía de un solo carril dividida en celdas discretas. Cada celda i en el tiempo t, denotada como $C_t(i)$, puede adoptar uno de dos estados binarios:

- **1 (Ocupado):** Contiene un vehículo.
- **0 (Vacío):** Espacio libre.

Definición Formal

La dinámica del sistema se rige por una lógica de evasión de colisiones. En cada paso temporal, todos los vehículos intentan avanzar simultáneamente una celda hacia la derecha. El movimiento solo ocurre si la celda destino está vacía.

Formalmente, la función de transición local δ que determina el estado $C_{t+1}(i)$ depende de la celda misma y de sus vecinos inmediatos:

$$C_{t+1}(i) = \begin{cases} 1, & \text{si } C_t(i-1) = 1 \wedge C_t(i) = 0 \quad \text{(llegada de un vehículo)}, \\ 1, & \text{si } C_t(i) = 1 \wedge C_t(i+1) = 1 \quad \text{(bloqueo por tráfico)}, \\ 0, & \text{en cualquier otro caso.} \end{cases}$$

Esta regla cumple estrictamente la propiedad de **conservación del número de partículas**, lo que significa que los vehículos no se crean ni se destruyen durante la simulación, sino que simplemente se desplazan o se detienen (Martínez et al., 2013).

Análisis Microscópico y Origen del Nombre

Para comprender explícitamente el comportamiento del algoritmo, se analizan las 8 configuraciones posibles de la vecindad local $\{C_t(i-1), C_t(i), C_t(i+1)\}$. La siguiente tabla de verdad detalla la interpretación física de cada transición:

Cuadro 2.3: Tabla de Transición de la Regla 184

Izq $(i-1)$	**Centro** (i)	**Der** $(i+1)$	**Nuevo** $C_{t+1}(i)$	**Interpretación Física**
1	1	1	**1**	Tráfico detenido: vehículo bloqueado.
1	1	0	**0**	Flujo libre: el vehículo avanza (deja hueco).
1	0	1	**1**	Avance: vehículo llega desde la izquierda.
1	0	0	**1**	Avance: vehículo llega desde la izquierda.
0	1	1	**1**	Bloqueo: obstáculo adelante impide moverse.
0	1	0	**0**	Movimiento: el vehículo se va a la derecha.
0	0	1	**0**	Vacío estático.
0	0	0	**0**	Vacío total.

El nombre Regla 184 deriva de la lectura de la columna Nuevo como un número binario. El patrón de bits resultante, ordenado desde la configuración $(1, 1, 1)$ hasta $(0, 0, 0)$, es 10111000_2, cuyo valor decimal es **184**.

Dinámica de Fluidos y Fases de Tráfico

Desde una perspectiva macroscópica, la Regla 184 actúa como el análogo discreto de la ecuación de continuidad en dinámica de fluidos (Ecuación de Burgers). El sistema exhibe una transición de fase crítica dependiente de la densidad de vehículos:

- **Fase de Flujo Libre :** Los vehículos se mueven a velocidad máxima sin interacciones negativas. El flujo es controlado por la demanda.
- **Fase de Congestión :** Emergen ondas cinemáticas de parada y arranque (*stop-and-go waves*) que se propagan hacia atrás, en sentido contrario al flujo vehicular.

2.8.2. Conservación de número y transición de fase libre–congestionado

En este modelo, el comportamiento colectivo del flujo vehicular depende de la densidad inicial ρ (número de vehículos dividido entre la longitud de la vía). Cuando $\rho < 0{,}5$, todos los vehículos pueden avanzar libremente, describiendo la **fase libre**. Cuando $\rho > 0{,}5$, surgen bloqueos que impiden el movimiento continuo, generando la **fase congestionada** (Boccara et al., 1993).

La transición entre ambas fases representa un ejemplo de **autoorganización crítica**: a partir de interacciones locales deterministas, el sistema global exhibe un cambio abrupto en su comportamiento colectivo. Según Adamatzky (2010) y Bar-Yam (1997), esta transición es análoga a los fenómenos de percolación y de fase en sistemas físicos complejos.

2.8.3. Modelos derivados

La Regla 184 ha inspirado una familia de modelos bidimensionales y estocásticos que amplían su aplicabilidad a contextos urbanos y de tráfico real:

- **Modelo BML (Biham–Middleton–Levine):** simula el tráfico en una cuadrícula bidimensional donde los vehículos avanzan por turnos en direcciones ortogonales (Biham et al., 1992). Presenta transiciones de fase espontáneas entre flujos libres y bloqueados.
- **Modelo NaSch (Nagel–Schreckenberg):** introduce velocidades múltiples, aleatoriedad y frenado probabilístico, permitiendo reproducir fenómenos de atasco y ondas de congestión (Nagel & Schreckenberg, 1992).
- **Modelos híbridos:** combinan reglas deterministas y estocásticas para representar la variabilidad del comportamiento humano y del entorno urbano (Martínez & Adamatzky, 2013).

- **Modelos de tráfico distribuido:** aplican redes de autómatas interconectados, donde cada nodo representa una intersección con decisiones probabilísticas de giro y prioridad (Adamatzky et al., 2008).

Estos modelos conforman una jerarquía de simulación computacional que va desde el flujo lineal hasta redes complejas con control local y sincronización adaptativa.

2.8.4. Caminatas Aleatorias en Redes de Tráfico

Si bien la Regla 184 es determinista en tramos rectos, el comportamiento global de los agentes en una red urbana se modela matemáticamente como una **Caminata Aleatoria sobre Grafos (Random Walk on Graphs)**.

En este proyecto, cada vehículo actúa como una partícula que realiza una caminata aleatoria dirigida. La trayectoria del agente A se define como una secuencia de nodos visitados $S = \{n_1, n_2, ..., n_k\}$, donde la transición entre el nodo i y el nodo j es un proceso estocástico gobernado por una matriz de probabilidad de transición P_{ij}.

$$P(n_{t+1} = j | n_t = i) = p_{ij}$$

Donde p_{ij} corresponde a la probabilidad de giro asignada a cada intersección en el modelo. De esta forma, el sistema integra la dinámica determinista de flujo (Regla 184) con la dinámica estocástica de trayectoria (Caminata Aleatoria), cumpliendo con el objetivo de simular la movilidad de individuos bajo principios de incertidumbre y sistemas complejos.

2.8.5. Simulación del tráfico urbano mediante autómatas

Los autómatas celulares ofrecen una alternativa eficiente a los modelos diferenciales clásicos debido a su naturaleza discreta, paralelizable y local. En la simulación urbana, cada celda puede representar un segmento de calle o un carril, mientras que las reglas definen la interacción entre vehículos y semáforos, giros, bloqueos y prioridades (Wolfram, 2002; Adamatzky, 2005).

Martínez et al. (2013) demuestran que los AC permiten replicar fenómenos de tráfico como la formación de embotellamientos, ondas de parada y recuperación de flujo. Además, su implementación es compatible con arquitecturas de cómputo de alto rendimiento (HPC), lo que posibilita la simulación de miles de vehículos y nodos en tiempo real.

2.8.6. Aplicación del modelo a Teposcolula (Oaxaca)

En la presente investigación, la Regla 184 y sus extensiones se aplican para modelar el tráfico en la red vial de San Pedro y San Pablo Teposcolula, Oaxaca. Se construyó una digitalización de la red urbana basada en coordenadas cartesianas discretizadas (*grid* 240 $\times$ 240), donde cada celda representa un tramo de calle con sentido, carriles y probabilidades de giro asociadas a los nodos de intersección.

Cada nodo se define como un conjunto de coordenadas con atributos:

$$N = \{(x, y), p_{\text{norte}}, p_{\text{sur}}, p_{\text{este}}, p_{\text{oeste}}\},$$

donde p_i representa la probabilidad de que un vehículo elija una dirección específica. Este enfoque permite modelar tanto el flujo local (microscópico) como los patrones globales de congestión.

Los escenarios adaptativos se simulan mediante variaciones dinámicas en la densidad inicial, el número de carriles y las condiciones de frontera. Se integran, además, mecanismos de control local y redistribución de flujo inspirados en la retroalimentación no lineal de Mainzer & Chua (2012), donde la prioridad o cierre temporal de calles se ajusta según la entropía y la saturación de los nodos.

2.8.7. Métricas de evaluación

Para cuantificar el comportamiento del sistema, se utilizan métricas derivadas de la teoría del tráfico y de los sistemas complejos (Bar-Yam, 1997; Nagel & Schreckenberg, 1992):

- **Flujo promedio (J):** número de vehículos que atraviesan una sección por unidad de tiempo.
- **Densidad (ρ):** proporción de celdas ocupadas respecto al total.
- **Congestión (C):** medida de la pérdida de movilidad, definida como $C = 1 - (J/J_{\text{máx}})$.
- **Tiempo medio de viaje (T_m):** promedio de pasos requeridos por vehículo desde la entrada hasta la salida.
- **Entropía de Shannon (H):** indicador del grado de desorden o variabilidad en la distribución de vehículos.

Estas métricas permiten evaluar el desempeño de distintas estrategias de control, como el ajuste probabilístico de giros o la modificación dinámica de prioridades, constituyendo un entorno experimental reproducible para validar el control adaptativo en redes de tráfico complejas.

2.9. Computación de alto rendimiento (HPC)

2.9.1. Concepto de HPC y su importancia en simulaciones complejas

La **Computación de Alto Rendimiento** (*High Performance Computing*, HPC) se refiere al uso coordinado de múltiples unidades de procesamiento (CPU o GPU) para resolver problemas computacionalmente intensivos mediante paralelización. En el estudio de sistemas complejos y autómatas celulares, el uso de HPC resulta esencial para analizar fenómenos emergentes a gran escala, donde las interacciones entre millones de celdas requieren tiempos de ejecución significativos (Dongarra & Sterling, 2017).

Bar-Yam (1997) señala que los sistemas complejos poseen escalas jerárquicas de interacción que sólo pueden capturarse mediante simulaciones paralelas, permitiendo observar patrones globales que emergen de reglas locales. Wolfram (2002) también

enfatiza que la exploración masiva de autómatas celulares constituye un experimento computacional en sí mismo, donde la potencia de cómputo permite descubrir leyes de comportamiento antes inalcanzables.

> "High performance computing extends the reach of experimental mathematics into the computational universe." (Wolfram, 2002, p. 1058)

2.9.2. Paradigmas de paralelización empleados

Para la simulación de TeposIA, se optó por un enfoque de **Paralelismo de Memoria Compartida (SMP)**, el cual es superior a los clústeres distribuidos (MPI) en contextos donde la latencia de memoria es crítica. Se implementaron tres niveles de optimización:

- **Nivel de Instrucción (JIT):** Utilizando la librería *Numba*, se compilaron las funciones críticas de geometría (algoritmo de Bresenham y rotación de vectores) a código máquina nativo, eliminando la sobrecarga del intérprete de Python y permitiendo el uso de instrucciones vectoriales del CPU (SIMD).
- **Nivel de Proceso (Task Parallelism):** Mediante la librería *multiprocessing*, el Algoritmo Genético distribuye la evaluación de escenarios completos entre todos los núcleos lógicos disponibles en el procesador, logrando una saturación del 100 % de la capacidad de cómputo (paralelismo vergonzosamente paralelo).
- **Aceleración Heterogénea (GPU):** Se delegó exclusivamente la inferencia de la Red Neuronal Predictiva a la GPU mediante *TensorFlow/CUDA*, creando una arquitectura híbrida donde la CPU maneja la física y la GPU maneja la ïnteligencia".

2.9.3. Optimización de autómatas celulares para ejecución masiva

La optimización de AC en entornos HPC requiere adaptar la lógica de actualización a operaciones vectorizadas. En lugar de iterar celda por celda, se utilizan operaciones matriciales que actúan sobre regiones completas del espacio. Adamatzky (2010) y Martínez et al. (2013) destacan que la naturaleza local de los autómatas los hace altamente paralelizables, ya que la evolución de cada celda depende únicamente de un vecindario pequeño.

Técnicas empleadas incluyen:

- Almacenamiento en arreglos *NumPy* o *CuPy* para cálculos vectorizados.
- Reducción de accesos a memoria mediante máscaras booleanas.
- Actualización asincrónica por bloques para minimizar la latencia.
- Renderizado incremental (*partial update*) para mejorar el rendimiento visual.

Estas optimizaciones permiten alcanzar velocidades del orden de 10^8 celdas/segundo en hardware moderno, dependiendo del tamaño del grid y la complejidad de la regla.

2.9.4. Stack Tecnológico de Alto Rendimiento

La implementación de TeposIA se basa en un stack científico optimizado para HPC en Python:

- **Numba (JIT Compiler):** Se utiliza para decorar funciones críticas (`@jit(nopython=True)`), permitiendo que el bucle principal de simulación alcance velocidades cercanas a C++ mediante compilación LLVM en tiempo de ejecución.
- **Multiprocessing Pool:** Gestiona un conjunto de procesos trabajadores (workers) aislados que ejecutan simulaciones independientes en paralelo, esencial para la convergencia rápida del Optimizador Genético.
- **NumPy (Estructuras de Datos):** Se implementaron matrices estáticas de enteros de 8 bits (`int8`) para el mapa de colisiones, optimizando el uso de la memoria caché del procesador y reduciendo la complejidad de acceso de $O(N)$ a $O(1)$.

2.9.5. Escalabilidad y eficiencia computacional

La **escalabilidad** mide la capacidad de un sistema para mantener o mejorar su rendimiento al incrementar los recursos computacionales. Se define como la relación entre el tiempo de ejecución T_n con n procesadores respecto al tiempo secuencial T_1:

$$E(n) = \frac{T_1}{n \cdot T_n}.$$

Un sistema se considera eficientemente escalable si $E(n) \approx 1$. En las simulaciones de tráfico, esta propiedad permite ampliar el tamaño del mapa o la cantidad de vehículos sin degradar el rendimiento perceptible (FPS) ni comprometer la sincronización entre celdas.

La eficiencia depende de la división espacial del dominio, la latencia de comunicación entre subprocesos y la carga computacional por bloque. Modelos como FLAME GPU 2 logran escalabilidad casi lineal hasta miles de núcleos CUDA (Richmond et al., 2021).

2.9.6. Elección del lenguaje de programación

El lenguaje **Python 3.10** se selecciona por su equilibrio entre claridad sintáctica, ecosistema científico y compatibilidad con entornos de HPC. Aunque lenguajes como C++ o Fortran ofrecen mayor velocidad nativa, Python cuenta con bibliotecas que acceden directamente a rutinas optimizadas en C, CUBLAS o MKL, garantizando alto rendimiento en operaciones vectorizadas (Oliphant, 2006).

La modularidad de Python facilita además la integración de interfaces gráficas (Tkinter, Pygame) con procesamiento paralelo, lo que permite combinar simulación científica con visualización interactiva en tiempo real.

2.9.7. Bibliotecas científicas y gráficas

Para la implementación práctica de los autómatas celulares y la visualización de resultados, se emplean las siguientes bibliotecas:

- **NumPy:** operaciones numéricas de alto rendimiento sobre arreglos multidimensionales.
- **Matplotlib:** generación de gráficos científicos y análisis de series temporales.
- **Tkinter:** interfaz gráfica ligera para control de simulaciones y paneles de parámetros.
- **Pygame:** renderizado interactivo de mapas, agentes y animaciones en tiempo real.

Estas herramientas conforman el entorno experimental del simulador **TeposIA v10–v11**, donde se combinan la computación paralela, la medición de entropía y la representación dinámica del tráfico urbano.

2.9.8. Integración con hardware GPU NVIDIA

El entorno de pruebas utiliza arquitecturas **NVIDIA Ampere** y **Ada Lovelace**, cuyas GPU (RTX 30xx–40xx) incorporan núcleos CUDA, Tensor y RT, optimizados para cómputo masivo. Estas unidades permiten procesar matrices de gran tamaño con mínima latencia, aprovechando la memoria GDDR6X y la ejecución concurrente de kernels.

El uso de librerías como CuPy o Numba CUDA permite ejecutar funciones Python directamente en GPU, alcanzando aceleraciones de hasta 50× respecto a CPU. En contextos de simulación urbana, esto posibilita la representación en tiempo real de mallas de 240×240 o superiores, con múltiples tipos de vehículos y métricas dinámicas.

2.9.9. Consideraciones de rendimiento y perfilado

El análisis del rendimiento se basa en indicadores clave:

- **Celdas/segundo:** cantidad de celdas actualizadas por ciclo de simulación.
- **FPS (frames per second):** velocidad de visualización gráfica durante la ejecución.
- **Uso de VRAM:** cantidad de memoria gráfica utilizada para almacenar los estados y texturas del autómata.

Herramientas como *NVIDIA Nsight*, *cProfile* y *line_profiler* permiten identificar cuellos de botella en la ejecución. La combinación de estas métricas proporciona una visión integral de la eficiencia del sistema, garantizando que la simulación mantenga un equilibrio entre precisión, estabilidad y velocidad en escenarios urbanos complejos.

2.10. Computación evolutiva: algoritmos genéticos

2.10.1. Fundamentos biológicos (Holland, 1975)

Los **algoritmos genéticos** (GA, por sus siglas en inglés) se inspiran en los procesos de la evolución natural descritos por Darwin, aplicando los principios de selección, reproducción y mutación al ámbito computacional. Fueron formalizados por John Holland en su obra seminal *Adaptation in Natural and Artificial Systems* (1975), donde propuso que la evolución biológica puede considerarse un proceso de búsqueda adaptativa en un espacio de soluciones.

Holland (1975) estableció que las soluciones a un problema pueden codificarse como cromosomas —secuencias de genes que representan parámetros o decisiones—, y que su combinación mediante operadores genéticos permite mejorar progresivamente el desempeño de la población. El enfoque se convirtió en la base de la **computación evolutiva**, un campo que abarca técnicas como los algoritmos genéticos, la programación genética, la evolución diferencial y las estrategias evolutivas (Back, Fogel, & Michalewicz, 2000).

> "Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics." (Holland, 1975, p. 3)

2.10.2. Conceptos clave

Un algoritmo genético está compuesto por los siguientes elementos fundamentales:

- **Codificación del cromosoma:** representa cada solución posible (individuo) mediante una cadena de genes binarios, enteros o reales. Cada gen corresponde a un parámetro del problema (Mitchell, 1996).
- **Función de aptitud (fitness):** evalúa la calidad o desempeño de cada individuo en relación con el objetivo de optimización. Puede definirse como una función $f(x)$ que mide la eficiencia, error o estabilidad de la solución.
- **Selección:** determina qué individuos se reproducen basándose en su aptitud. Los métodos más comunes incluyen la ruleta (*roulette wheel*) y el torneo (*tournament selection*).
- **Cruce (crossover):** combina pares de cromosomas para generar descendencia, intercambiando segmentos genéticos. Ejemplo: cruce de un punto o uniforme.
- **Mutación:** altera aleatoriamente algunos genes para introducir diversidad y evitar la convergencia prematura.

El proceso iterativo de selección, cruce y mutación continúa hasta alcanzar una condición de parada, generalmente basada en el número de generaciones o en la estabilidad de la función de aptitud.

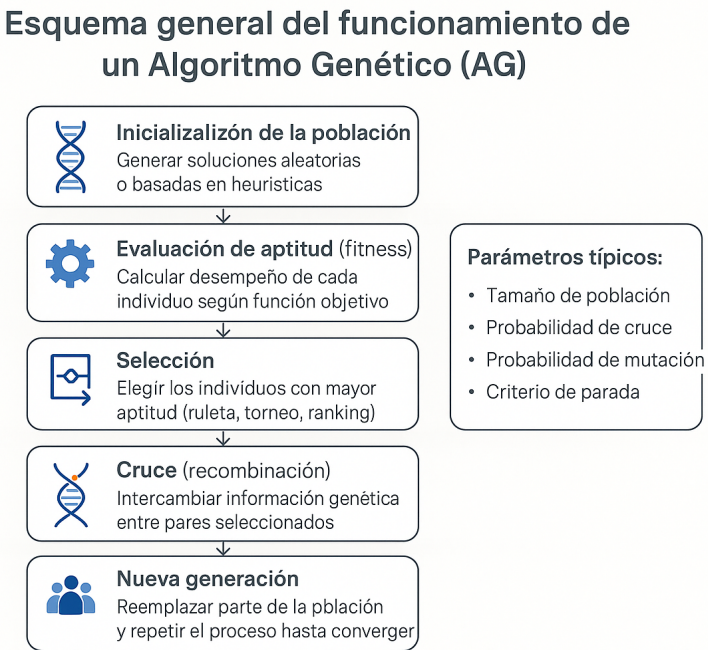


Figura 2.1: Esquema general del funcionamiento de un algoritmo genético.

2.10.3. Variantes de algoritmos evolutivos

A partir del paradigma genético propuesto por Holland, surgieron diversas variantes especializadas de computación evolutiva:

- **GA (Genetic Algorithm):** enfoque clásico basado en cadenas binarias o reales, ampliamente aplicado en optimización combinatoria (Mitchell, 1996).
- **GP (Genetic Programming):** propuesta por Koza (1992), evoluciona programas o expresiones simbólicas completas en lugar de vectores de parámetros.
- **DE (Differential Evolution):** introducida por Storn y Price (1997), utiliza diferencias entre individuos para generar variación, especialmente eficaz en espacios continuos.
- **ES (Evolution Strategies):** propuesta por Rechenberg (1973) y Schwefel (1995), orientada a la optimización de funciones reales mediante mutaciones gaussianas.

Estas variantes comparten principios comunes —poblaciones, operadores estocásticos y adaptación iterativa—, pero difieren en la representación, la dinámica de búsqueda y la presión selectiva aplicada a las soluciones.

2.10.4. Aplicación en sistemas complejos

Los algoritmos genéticos se aplican ampliamente en el estudio y control de sistemas complejos debido a su capacidad de explorar espacios de búsqueda no lineales y multimodales. En particular, ofrecen ventajas para la optimización de parámetros, el ajuste de reglas locales y la calibración adaptativa de modelos emergentes.

Entre sus aplicaciones más relevantes se incluyen:

- **Optimización de parámetros de simulación:** ajuste de densidad, probabilidad de giro, y reglas de prioridad en modelos de tráfico basados en autómatas (Martínez & Adamatzky, 2013).
- **Evolución de configuraciones iniciales:** selección de estados iniciales en autómatas celulares para generar patrones deseados o maximizar la diversidad estructural (Adamatzky, 2010).
- **Ajuste de reglas locales:** exploración del espacio de reglas (por ejemplo, B/S en modelos tipo Life) para obtener comportamientos específicos de autoorganización o flujo óptimo (Wolfram, 2002).
- **Calibración de controladores de tráfico:** evolución de estrategias de control descentralizado basadas en entropía, densidad y flujo vehicular en nodos de intersección (Bar-Yam, 1997; Nagel & Schreckenberg, 1992).

En sistemas no lineales y adaptativos, los algoritmos evolutivos permiten encontrar soluciones robustas incluso bajo incertidumbre o condiciones cambiantes, lo que los convierte en una herramienta ideal para la simulación urbana y la optimización del flujo vehicular.

2.10.5. Implementación en Python

Python ofrece múltiples bibliotecas para la implementación de algoritmos genéticos y evolutivos, integrándose fácilmente con entornos científicos e interfaces gráficas. Entre las más destacadas se encuentran:

- **DEAP (Distributed Evolutionary Algorithms in Python):** biblioteca modular que soporta GA, GP, ES y DE, con soporte para paralelización mediante *multiprocessing* (Fortin et al., 2012).
- **PyGAD:** framework liviano para optimización mediante GA, diseñado para integrarse con redes neuronales, entornos NumPy y simulaciones dinámicas (Gad, 2021).

Un esquema genérico en Python utilizando `PyGAD` puede resumirse como:

```
import pygad, numpy as np

def fitness_func(solution, solution_idx):
    return -abs(np.sum(solution) - target_value)

ga = pygad.GA(num_generations=100,
              num_parents_mating=10,
              fitness_func=fitness_func,
              sol_per_pop=20,
              num_genes=6,
              mutation_percent_genes=10)

ga.run()
ga.plot_result()
```

Este tipo de implementación permite integrar la optimización evolutiva directamente con las métricas del simulador (flujo, densidad, entropía), posibilitando la evolución automática de escenarios de tráfico.

2.10.6. Evaluación de convergencia y robustez

La evaluación de un algoritmo genético implica medir su capacidad para converger hacia soluciones óptimas sin perder diversidad poblacional. La **convergencia** se analiza mediante el seguimiento de la mejor y la media de las aptitudes por generación, mientras que la **robustez** se evalúa frente a perturbaciones, ruido o variaciones en las condiciones iniciales (Back et al., 2000).

La diversidad genética se mantiene mediante una tasa de mutación adaptativa o reinicialización parcial de la población. En sistemas complejos, la robustez evolutiva garantiza estabilidad ante entornos caóticos, lo cual es esencial para el control adaptativo de flujos vehiculares simulados.

> "Robustness in evolutionary computation is not resistance to change, but the capacity to adapt efficiently to it." (Mainzer & Chua, 2012, p. 167)

2.11. Redes neuronales y aprendizaje en sistemas complejos

2.11.1. Principios de las redes neuronales artificiales

Las **redes neuronales artificiales** (RNA) son modelos computacionales inspirados en el funcionamiento del cerebro humano, formados por unidades simples llamadas *neuronas*, conectadas entre sí mediante pesos sinápticos que se ajustan a partir de la experiencia. McCulloch y Pitts (1943) introdujeron el primer modelo formal de neurona lógica, mientras que Rosenblatt (1958) desarrolló el *perceptrón*, precursor del aprendizaje supervisado.

Según Haykin (1999), las RNA constituyen sistemas adaptativos que aprenden a partir de datos mediante la modificación iterativa de sus conexiones internas. Esta capacidad las convierte en herramientas idóneas para modelar y controlar sistemas no lineales, caóticos o de alta dimensionalidad, propios de los sistemas complejos.

> "An artificial neural network is a massively parallel distributed processor that learns from experience and stores knowledge in its connections." (Haykin, 1999, p. 2)

2.11.2. Arquitecturas

Las arquitecturas de redes neuronales se clasifican según la dirección del flujo de información y la organización de sus capas. Entre las más relevantes destacan:

- **Perceptrón Multicapa (MLP):** red feedforward con una o más capas ocultas, capaz de aproximar funciones continuas y no lineales (Rumelhart, Hinton, & Williams, 1986). Utiliza funciones de activación como ReLU, sigmoide o tangente hiperbólica.

- **Redes Convolucionales (CNN):** diseñadas para procesar datos espaciales mediante filtros que extraen características locales. Son ampliamente utilizadas en análisis de imágenes, mapas o patrones espaciotemporales (LeCun et al., 1998).
- **Redes Recurrentes (RNN):** incluyen conexiones de retroalimentación que permiten capturar dependencias temporales en secuencias. Las variantes LSTM (*Long Short-Term Memory*) y GRU (*Gated Recurrent Unit*) resuelven el problema del desvanecimiento del gradiente (Hochreiter & Schmidhuber, 1997).

Estas arquitecturas proporcionan distintas formas de representar la dinámica de los sistemas complejos: las MLP aprenden relaciones no lineales, las CNN extraen estructuras espaciales y las RNN modelan dependencias temporales y secuenciales.

2.11.3. Entrenamiento y ajuste de pesos

El proceso de aprendizaje de una red neuronal se realiza mediante el algoritmo de **retropropagación del error** (Rumelhart et al., 1986), que ajusta los pesos w_{ij} minimizando una función de pérdida E mediante gradiente descendente:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \eta \frac{\partial E}{\partial w_{ij}},$$

donde η es la tasa de aprendizaje.

Este procedimiento permite que la red mejore su rendimiento iterativamente, reduciendo la discrepancia entre la salida estimada y la salida deseada. En contextos no lineales, como la predicción de densidad vehicular o la clasificación de estados de tráfico, la retropropagación se combina con normalización de datos, regularización y técnicas de optimización avanzadas (Adam, RMSprop) para evitar el sobreajuste y acelerar la convergencia (Goodfellow, Bengio, & Courville, 2016).

2.11.4. Aplicaciones en control y predicción

Las redes neuronales han demostrado una gran capacidad para modelar, predecir y controlar sistemas complejos caracterizados por relaciones no lineales y comportamiento emergente. En el ámbito del tráfico vehicular, sus aplicaciones más relevantes incluyen:

- **Predicción de densidad y flujo vehicular:** las RNA pueden anticipar la congestión a partir de datos históricos y sensores simulados, optimizando la toma de decisiones (Lv et al., 2015).
- **Clasificación de estados de tráfico:** uso de CNN y MLP para categorizar el sistema en estados *fluido*, *crítico* o *congestionado* mediante análisis de patrones de densidad y entropía (Martínez et al., 2013).
- **Control adaptativo:** integración de redes neuronales en sistemas de retroalimentación para ajustar parámetros de semáforos o probabilidades de giro, mejorando la eficiencia global (Bar-Yam, 1997; Mainzer & Chua, 2012).

Estas técnicas permiten construir controladores capaces de aprender en tiempo real, adaptándose a las fluctuaciones del sistema y a las condiciones dinámicas de la red vial.

2.11.5. Reservoir Computing

El paradigma de **Reservoir Computing** combina las ventajas de las redes recurrentes con la simplicidad del entrenamiento lineal. En este modelo, la dinámica interna del sistema actúa como un *reservorio* de estados que transforma las entradas en una representación de alta dimensión, mientras que sólo la capa de salida se entrena linealmente (Jaeger, 2001).

Dos de sus principales implementaciones son:

- **Echo State Networks (ESN):** emplean un conjunto de neuronas recurrentes aleatorias con conexiones fijas. El entrenamiento sólo ajusta los pesos de salida, garantizando estabilidad mediante la propiedad de *eco* (Jaeger, 2001).
- **Liquid State Machines (LSM):** inspiradas en neurociencia, utilizan neuronas impulsadas por picos (*spiking*) que generan una respuesta temporal rica, permitiendo procesar señales dinámicas (Maass, Natschläger, & Markram, 2002).

En el contexto de sistemas complejos, el Reservoir Computing ofrece una forma eficiente de capturar la dinámica temporal sin necesidad de entrenamiento profundo, siendo útil para el control adaptativo y la predicción de variables de tráfico en entornos HPC.

2.11.6. Relación entre autómatas celulares y redes neuronales

Aunque los autómatas celulares (AC) y las redes neuronales pertenecen a paradigmas distintos, ambos comparten fundamentos comunes: localismo, paralelismo y aprendizaje distribuido. Wolfram (2002) y Adamatzky (2010) sostienen que los AC pueden considerarse redes neuronales discretas, donde las reglas de transición cumplen el papel de funciones de activación, y el estado de cada celda equivale a la salida de una neurona.

Investigaciones recientes (Gilpin, 2019) demuestran que es posible entrenar redes neuronales para aprender las reglas de un autómata celular (*Neural Cellular Automata*), lo que fusiona los dos marcos de modelado: el aprendizaje continuo y la dinámica discreta emergente. Esto abre la puerta a simuladores inteligentes donde las reglas de evolución se ajustan automáticamente según las condiciones del entorno, en lugar de estar fijadas a priori.

> "Neural Cellular Automata blur the boundary between designed rules and learned behaviors, merging evolution with learning." (Gilpin, 2019)

2.11.7. Integración en el sistema TeposIA

En el futuro del proyecto **TeposIA**, las redes neuronales se integran como módulo de aprendizaje y predicción para mejorar la eficiencia y adaptabilidad del sistema de simulación de tráfico urbano. El objetivo es que la red aprenda patrones de flujo y congestión a partir de los datos generados por la simulación basada en la Regla 184 y sus extensiones.

Las aplicaciones específicas incluyen:

- **Aprendizaje de patrones dinámicos:** el sistema neuronal identifica configuraciones recurrentes en la red vial (nodos críticos, rutas congestionadas).
- **Predicción adaptativa:** modelos MLP o LSTM anticipan el comportamiento del tráfico en función del tiempo y la densidad local.
- **Control inteligente:** el sistema ajusta en tiempo real las probabilidades de giro o apertura de calles, minimizando la entropía global y el tiempo medio de viaje.

En versiones avanzadas del simulador, se plantea el uso de arquitecturas híbridas AC–RNA donde la red neuronal evoluciona los parámetros de control del autómata (densidad, velocidad, prioridad), estableciendo un esquema de **aprendizaje emergente** que combina autoorganización con inteligencia artificial.

2.12. Integración de control inteligente en medios no lineales

2.12.1. Convergencia entre modelado discreto, HPC e IA

El control inteligente en medios no lineales surge de la convergencia entre tres enfoques complementarios: el **modelado discreto** mediante autómatas celulares (AC), la **computación de alto rendimiento** (HPC) para la ejecución masiva de simulaciones, y la **inteligencia artificial** (IA) como mecanismo de aprendizaje y adaptación. Mainzer y Chua (2012) describen esta integración como una manifestación del principio de *actividad local*, donde la complejidad se regula mediante realimentaciones jerárquicas y autoorganización controlada.

El modelado discreto permite capturar la dinámica local de los sistemas complejos, mientras que la HPC ofrece la capacidad de simular en paralelo millones de interacciones. Finalmente, la IA introduce un nivel superior de abstracción, capaz de aprender patrones, predecir comportamientos y ajustar las reglas de control en tiempo real.

Wolfram (2002) plantea que esta sinergia constituye una forma de exploración computacional del universo: las leyes simples (AC) se complementan con el aprendizaje emergente (IA) y el procesamiento masivo (HPC), generando entornos donde el orden y el caos coexisten bajo una estructura adaptativa.

> "The fusion of discrete rules, parallel computation, and learning algorithms represents the next step in exploring complex behavior by design." (Wolfram, 2002, p. 1043)

2.12.2. Arquitectura de control híbrido (AC + RNA + AG)

La arquitectura propuesta combina tres niveles de procesamiento complementarios:

1. **Nivel discreto (Autómata Celular, AC):** modela la dinámica física del sistema —en este caso, el tráfico urbano— mediante reglas locales deterministas (como la Regla 184 o sus extensiones BML/NaSch).

2. **Nivel de aprendizaje (Red Neuronal Artificial, RNA):** predice el comportamiento global (flujo, densidad, congestión) y ajusta los parámetros del AC (probabilidades de giro, velocidades, prioridades).

3. **Nivel evolutivo (Algoritmo Genético, AG):** optimiza los hiperparámetros del sistema de control y los pesos iniciales de la RNA, explorando soluciones robustas mediante evolución adaptativa.

Este enfoque híbrido AC–RNA–AG, inspirado en los principios de la computación natural (Adamatzky, 2010), crea un sistema de **retroalimentación inteligente**, donde la evolución, el aprendizaje y la simulación interactúan en ciclos iterativos. La HPC permite la ejecución paralela de múltiples escenarios, acelerando la búsqueda de configuraciones óptimas.

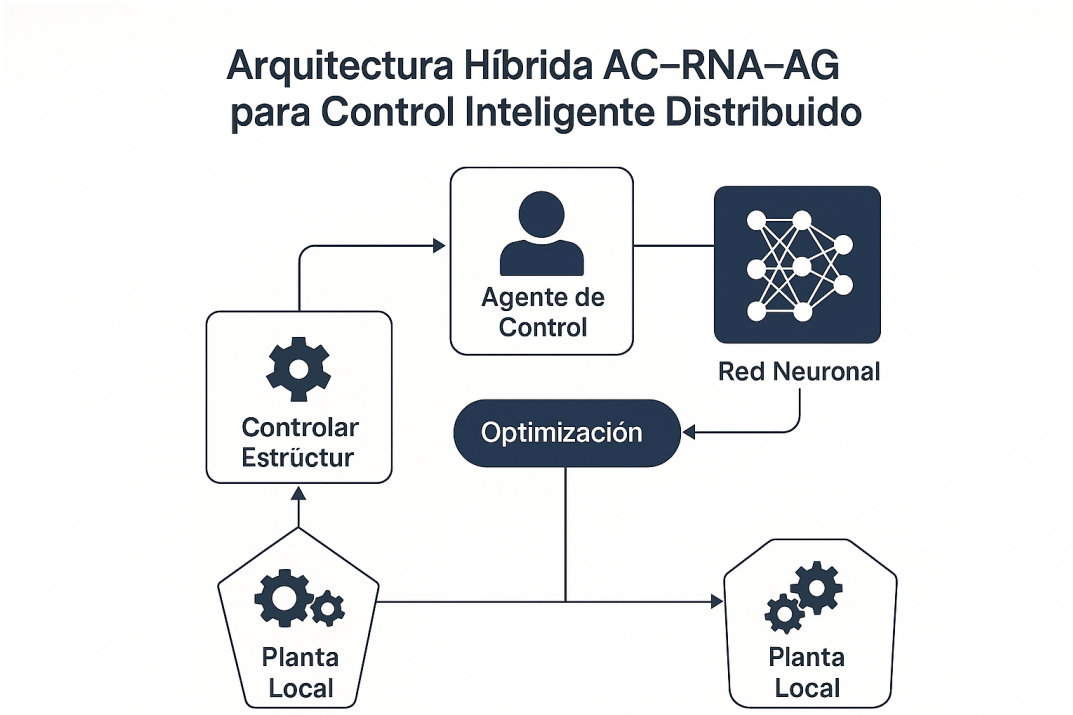


Figura 2.2: Arquitectura híbrida AC–RNA–AG para control inteligente distribuido.

2.12.3. Ciclo de simulación–adaptación–control

El funcionamiento del sistema integrado sigue un ciclo dinámico continuo:

1. **Simulación:** el autómata celular ejecuta la dinámica del tráfico bajo condiciones iniciales definidas.

2. **Medición:** se calculan métricas locales y globales (densidad, flujo, entropía, congestión).

3. **Adaptación:** la red neuronal analiza los patrones emergentes y predice el estado futuro del sistema.

4. **Control:** el algoritmo genético ajusta las reglas o probabilidades del AC para minimizar la entropía o mejorar el flujo.

5. **Reinicialización:** se actualizan los parámetros y se reinicia la simulación bajo nuevas condiciones optimizadas.

Este ciclo representa un proceso de autoaprendizaje distribuido en el que la simulación y el control coevolucionan. En el contexto de sistemas urbanos, permite desarrollar estrategias de control descentralizado donde cada nodo (intersección) aprende y adapta su comportamiento de forma local, contribuyendo a la estabilidad global.

2.12.4. Evaluación multiobjetivo

El control inteligente se evalúa mediante un enfoque **multiobjetivo**, donde el sistema busca simultáneamente optimizar varios criterios de desempeño. Entre los objetivos considerados destacan:

- **Eficiencia:** maximización del flujo vehicular J y minimización del tiempo medio de viaje T_m.
- **Estabilidad:** reducción de oscilaciones y congestión recurrente mediante la disminución de la entropía de Shannon H.
- **Adaptabilidad:** capacidad del sistema para responder ante perturbaciones externas o cambios en la densidad inicial.

El algoritmo genético define una función de aptitud compuesta:

$$F = \alpha(1 - C) + \beta\frac{J}{J_{\max}} + \gamma(1 - \frac{H}{H_{\max}}),$$

donde α, β y γ son coeficientes ponderados que representan la importancia relativa de cada criterio. Este enfoque permite equilibrar la eficiencia global con la estabilidad y la adaptabilidad del sistema.

2.12.5. Métricas de desempeño del control distribuido

Para cuantificar el comportamiento del sistema de control inteligente, se emplean métricas dinámicas derivadas de la teoría de sistemas complejos y del análisis de tráfico (Bar-Yam, 1997; Martínez et al., 2013):

- **Flujo global (J):** tasa promedio de vehículos que cruzan una sección.
- **Entropía (H):** mide el grado de desorden o incertidumbre en la distribución del tráfico.
- **Error de predicción (E_p):** diferencia entre el flujo real y el estimado por la RNA.
- **Índice de estabilidad (S):** cuantifica la variación temporal de la densidad y su convergencia.
- **Rendimiento HPC (R_{HPC}):** celdas actualizadas por segundo y eficiencia de paralelización.

Estas métricas se monitorean en tiempo real, permitiendo la evaluación simultánea del comportamiento físico, computacional y adaptativo del sistema.

2.12.6. Concepto de emergencia controlada

El término **emergencia controlada** describe la capacidad de un sistema complejo para generar patrones organizados sin perder estabilidad global. Mainzer y Chua (2012) argumentan que el control no debe eliminar la autoorganización, sino dirigirla hacia configuraciones deseadas. En este contexto, la retroalimentación adaptativa funciona como un "campo de atracción" que guía la emergencia sin suprimirla.

En el marco de la simulación, la emergencia controlada se manifiesta cuando las congestiones espontáneas son mitigadas por decisiones locales aprendidas, sin necesidad de una autoridad central. El sistema mantiene su diversidad y capacidad de adaptación, alcanzando un equilibrio dinámico entre orden y caos.

> "Controlled emergence represents the balance between freedom and regulation —a self-organizing intelligence." (Mainzer & Chua, 2012, p. 192)

2.12.7. Proyección hacia control neuromórfico y morfogenético

El futuro del control inteligente en medios no lineales se orienta hacia paradigmas inspirados en la biología y la neurociencia, como el **control neuromórfico** y el **control morfogenético**. El primero busca replicar los principios de procesamiento paralelo y plasticidad sináptica del cerebro en hardware especializado (Mead, 2020), mientras que el segundo se basa en la autoorganización de formas y estructuras, emulando procesos de desarrollo biológico (Adamatzky, 2005).

Estos enfoques convergen en el concepto de **computación morfogenética**, donde los sistemas físicos se adaptan a través de reglas locales aprendidas, combinando inteligencia artificial, dinámica no lineal y hardware reconfigurable. La aplicación de estos principios en simulaciones urbanas abre la posibilidad de sistemas de tráfico autoorganizados que aprenden, evolucionan y se regeneran ante cambios del entorno, marcando el paso hacia una nueva era de *inteligencia distribuida en medios físicos.*

Capítulo 3

DESARROLLO

3.0.1. Identificación y Análisis de Requerimientos

Para garantizar la viabilidad técnica y el alcance del proyecto, se definieron requerimientos detallados abarcando la simulación física, la inteligencia artificial y el rendimiento computacional.

Requerimientos Funcionales (RF)

Los requerimientos funcionales describen las interacciones específicas del sistema y su comportamiento esperado:

- **RF1. Motor de Micro-Simulación:** El sistema debe simular el flujo vehicular utilizando el modelo de autómata celular (Regla 184 extendida) garantizando la conservación de vehículos y la ausencia de colisiones lógicas.
- **RF2. Modelado de Red Vial:** Capacidad para cargar y representar mapas complejos desde archivos JSON, incluyendo múltiples carriles, sentidos de circulación y zonas urbanas definidas.
- **RF3. Intersecciones Probabilísticas:** Los nodos de cruce deben soportar matrices de decisión estocástica (probabilidades de giro) configurables individualmente para simular el comportamiento no determinista de los conductores..
- **RF4. Módulo de Optimización (Evolutivo):** Implementación de un Algoritmo Genético que evolucione los parámetros de los semáforos y probabilidades de giro para maximizar el flujo vehicular global.
- **RF5. Generación de Métricas y Reportes:** El sistema debe calcular y exportar en tiempo real métricas de entropía de Shannon, velocidad promedio, densidad y número de vehículos detenidos en formato CSV.
- **RF6. Interacción en Tiempo Real:** El usuario debe poder interactuar con la simulación en ejecución (agregar bloqueos, modificar semáforos, inyectar vehículos) sin detener el motor de procesamiento.

Requerimientos No Funcionales (RNF)

Los requerimientos no funcionales establecen los criterios de calidad y restricciones técnicas, con énfasis en el alto rendimiento (HPC):

- **RNF1. Eficiencia de Ejecución (HPC-JIT):** El núcleo de simulación debe utilizar compilación Just-In-Time (Numba) para asegurar un tiempo de actualización de celdas menor a 50ms en mallas de 240×240.
- **RNF2. Paralelismo de Procesos:** El módulo de optimización genética debe ser capaz de utilizar el 100 % de los núcleos lógicos disponibles en el CPU (Multiprocessing) para la evaluación simultánea de escenarios.
- **RNF3. Estructura de Datos Optimizada:** El uso de memoria debe optimizarse mediante arreglos tipados estáticos (NumPy `int8`) en lugar de listas dinámicas de objetos, garantizando escalabilidad.
- **RNF4. Usabilidad y Visualización:** La interfaz gráfica debe mantener una tasa de refresco mínima de 30 FPS, proporcionando visualización clara mediante mapas de calor y gráficos dinámicos.

3.0.2. Gestión del Proyecto mediante Metodología Kanban

Para la administración de las tareas de desarrollo, se implementó un tablero Kanban digital. Esta metodología ágil permitió visualizar el flujo de trabajo, limitar el trabajo en curso (WIP) y gestionar las prioridades de manera dinámica conforme evolucionaban los requerimientos del simulador.

Estructura del Flujo de Trabajo

El tablero se organizó en cinco columnas de estado, asegurando un ciclo de desarrollo continuo:

1. **Backlog (Pila de Producto):** Repositorio de todas las funcionalidades deseadas, mejoras teóricas e ideas de optimización (ej. Ïntegrar Numba", "Diseñar mapa JSON").
2. **To Do (Por hacer):** Tareas seleccionadas para la iteración actual, priorizadas según su impacto en la estabilidad del sistema.
3. **In Progress (En proceso):** Tareas activas. Se estableció un límite de WIP (*Work In Progress*) de 2 tareas simultáneas para garantizar la calidad del código y evitar la dispersión.
4. **Testing / Validation:** Fase crítica donde se verificaba que las nuevas implementaciones (ej. nuevos giros) no rompieran la lógica de la Regla 184 o la coherencia del tráfico.
5. **Done (Finalizado):** Tareas completadas, refactorizadas y fusionadas en la versión estable del simulador.

Registro de Actividades Clave

A continuación, se presenta un resumen de las actividades principales gestionadas a través del tablero Kanban, clasificadas por la fase de desarrollo:

Cuadro 3.1: Desglose de actividades principales en el tablero Kanban

Fase / Versión	**Actividad (Task)**	**Tipo**
Fase Inicial (V1-V3)	Implementación del motor base Regla 184 en matriz 1D	Backend
	Creación de la clase *SimulationEngine* y bucle principal	Arq. Software
Fase Intermedia (V4-V7)	Desarrollo del parser JSON para carga de mapas	Datos
	Implementación de intersecciones con probabilidad simple	Lógica
	Integración de interfaz gráfica con Tkinter y Canvas	UI
Fase Avanzada (V8-V10)	Refactorización para soporte de múltiples carriles (Flows)	Optimización
	Desarrollo del sistema de métricas (Entropía y Densidad)	Análisis
Fase Final (V11-V12)	**Implementación de HPC:** Integración de Numba y JIT	HPC
	Módulo IA: Entrenamiento e integración de Red Neuronal LSTM	Inteligencia Art.
	Módulo AG: Desarrollo del Optimizador Genético con Multiprocessing	HPC/Evolutivo

Esta organización permitió mantener un ritmo de desarrollo constante, facilitando la detección temprana de cuellos de botella en el rendimiento del simulador y permitiendo la integración incremental de tecnologías complejas como HPC e IA.

Requerimientos estructurales del JSON

El archivo JSON debía contener:

- Lista completa de calles con coordenadas discretas.
- Conjunto de nodos con probabilidades de giro definidas por dirección.
- Sentido de circulación, número de carriles y prioridades.
- Posible expansión con zonas, polígonos, restricciones y cierres.

Estos requerimientos guiaron la construcción de la arquitectura técnica del sistema.

3.1. Desarrollo Evolutivo del Simulador (V1–V12)

Una de las partes centrales del proyecto fue la creación de un conjunto de versiones evolutivas. Cada versión introduce características nuevas, soluciona problemas anteriores y responde a los fenómenos emergentes observados.

A continuación se documenta cada versión con: objetivo, características, problemas, resultados y mejoras próximas.

—

V1. Motor base — Regla 184 unidimensional

Objetivo: Implementar la regla de movimiento fundamental.
Características:

- Movimiento unidimensional conservativo.
- Sin interfaz gráfica; sólo validación de matriz.

Problemas detectados: saturación prematura en densidades >0.05
Resultados: modelo base validado según Wolfram (2002).
Mejoras próximas: visualización gráfica y transición a 2D.

—

V2. Visualización elemental

Objetivo: Mostrar el tráfico en tiempo real.
Características:

- Canvas inicial con celdas blancas y negras.
- Control básico de iteraciones.

Problemas: bajo rendimiento al crecer la matriz.
Resultados: simulación visible para experimentación.
Mejoras: optimización y soporte para calles reales.

—

V3. Primeras calles N–S y E–O

Objetivo: Modelar topología urbana mínima.
Características:

- Dos ejes cardinales simples.
- Separación de carriles.

Problemas: cars "fantasma" en bordes.
Resultados: estructura vial mínima estable.
Mejoras: intersecciones.

—

V4. Intersecciones simples

Objetivo: Añadir nodos de cruce.
Características:

- Cruces de 2 calles.
- Movimiento recto y giros a la derecha.

Problemas: colisiones lógicas en tiempos simultáneos.
Resultados: cruces funcionales deterministas.
Mejoras: probabilidades direccionales.

—

V5. Probabilidades globales

Objetivo: Controlar los giros.
Características:

- Distribución global: recto/derecha/izquierda.
- Configuración estática.

Problemas: comportamiento rígido.
Resultados: capacidad básica de decisión.
Mejoras: probabilidades por nodo.

—

V6. Primer JSON urbano

Objetivo: Cargar mapas desde archivo externo.
Características:

- Calles, sentidos, nodos básicos.
- Estructura inicial por listas.

Problemas: formato poco legible.
Resultados: mapa dinámico cargado con éxito.
Mejoras: JSON estructurado por línea.

—

V7. Mapa Tepos V7 — 240×240

Objetivo: Representar la topología real de Teposcolula.
Características:

- Calles reales: Guerrero, Juárez, Díaz, etc.
- Carriles realistas.
- Coordenadas discretizadas.

Problemas: aparición de gridlock.
Resultados: red urbana completa.
Mejoras: segmentación por zonas.

—

V8. Múltiples carriles y flujos simultáneos

Objetivo: Flujo realista por carril.
Características:

- Manejo independiente por carril.
- Movimientos paralelos.

Problemas: desbordes en giros.
Resultados: simulación fluida.
Mejoras: gestión de prioridades.

—

V9. Giros completos cardinales

Objetivo: Definir comportamiento completo en cruces.
Características:

- Desde_N, desde_S, desde_E, desde_O.
- Probabilidades locales por nodo.

Problemas: pesos no coincidentes (ValueError).
Resultados: movilidad realista.
Mejoras: validaciones automáticas.

—

V10. Interfaz y control avanzados

Objetivo: Manejo interactivo y visual.
Características:

- Zoom, arrastre, step-by-step.
- Panel de control lateral.

Problemas: errores Tkinter en callbacks.
Resultados: simulación totalmente operable.
Mejoras: indicadores de métricas.

—

V11. JSON Estándar por línea

Objetivo: Crear un formato escalable.
Características:

- Un nodo por línea.
- Probabilidades definidas con precisión.

Problemas: alineación con mapa previo.
Resultados: JSON listo para IA.
Mejoras: zonas inteligentes.

—

V12. Versión integrada final

Objetivo general: Integrar todo el ecosistema AC–UI–JSON en un único simulador estable.

Características implementadas:

- Motor completo basado en Regla 184 extendida.
- Mapa Tepos 240×240 con nodos probabilísticos.
- Giros cardinales, múltiples carriles y flujos independientes.
- Métricas de densidad, flujo, entropía y congestión.
- Interfaz avanzada con zoom, control de velocidad, pausar/reanudar.
- JSON perfectamente estructurado y escalable.

Problemas enfrentados:

- Congestión espontánea en nodos no balanceados.

Resultados:

- Simulador urbano funcional en tiempo real.
- Comportamientos emergentes reproducidos: ondas de choque, formación de colas, autoorganización.
- Entorno preparado para IA (RNA) y algoritmos evolutivos (AG).

Mejoras futuras:

- Convertir en una aplicación nativa.
- Integración de control neuronal predictivo.
- Optimización CUDA completa del motor AC.
- Sistema de control adaptativo por nodo.

3.1.1. Interfaz:

- Editor.
- Simulador.
- Optimizador.
- Predictor.

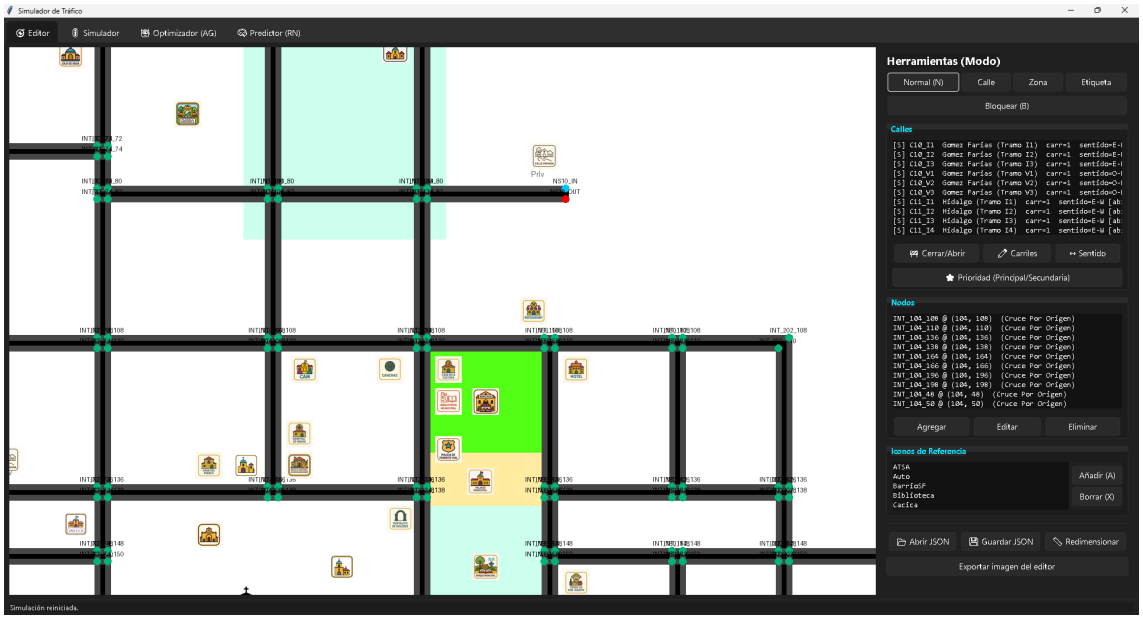


Figura 3.1: Interfaz: Pestaña editor

Figura 3.2: Interfaz: Pestaña editor

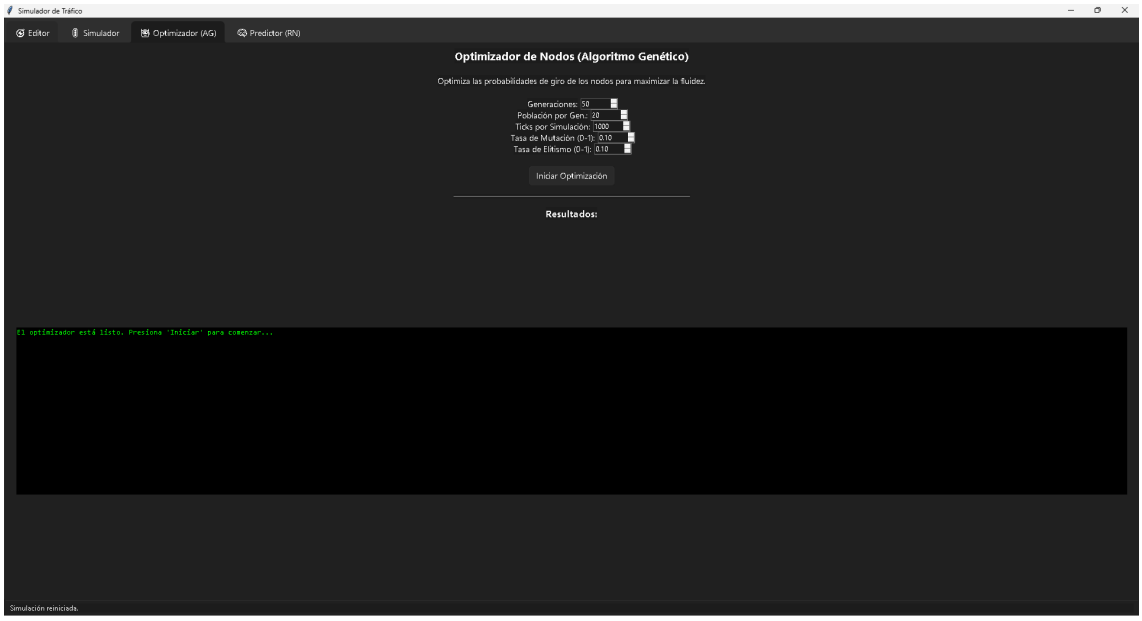


Figura 3.3: Interfaz: Pestaña optimizador

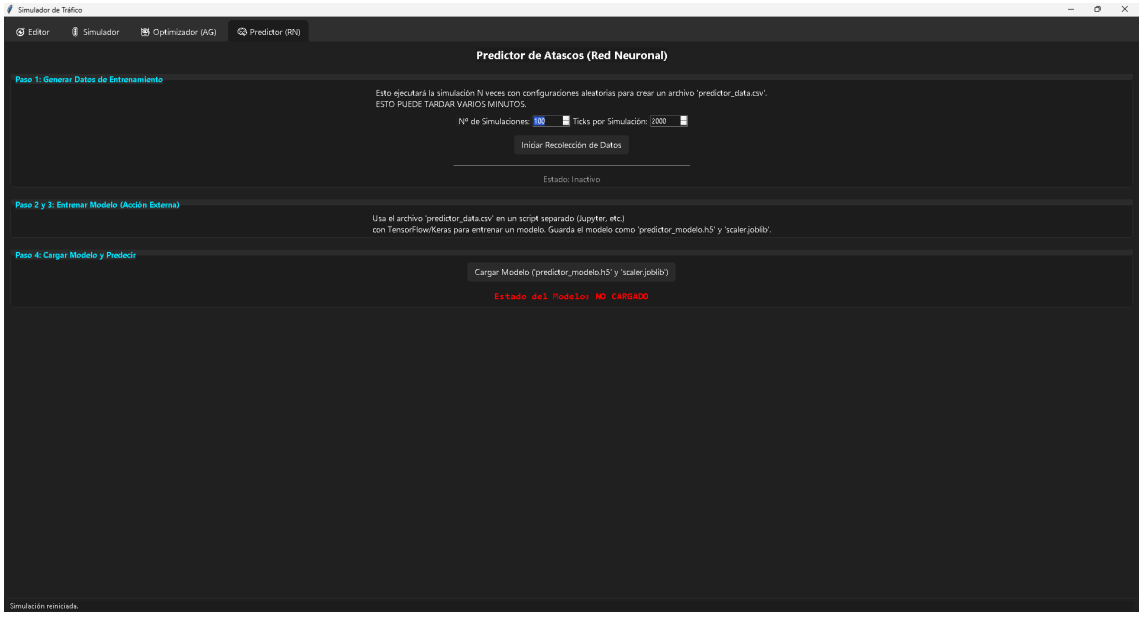


Figura 3.4: Interfaz: Pestaña predictor

3.2. Implementación de la Arquitectura Híbrida CPU-GPU

Uno de los desafíos principales fue la optimización del rendimiento para permitir la evolución genética en tiempos viables. La solución implementada en la versión final (`HPCc3`) divide la carga computacional en dos dominios de hardware:

3.2.1. Dominio de Física (CPU Optimizada)

El movimiento de los agentes (vehículos) es secuencial por naturaleza debido a las interacciones locales. Para optimizarlo, se abandonó el uso de objetos pesados de Python en favor de **arreglos primitivos de NumPy** y funciones compiladas con JIT (Just-In-Time) mediante la librería *Numba*.

> Se implementó una *matriz estática* (`static_grid`) de tipo `int8` que reside en memoria, permitiendo que el algoritmo de colisión consulte el estado de las calles con complejidad $O(1)$, eliminando la sobrecarga de iterar sobre listas de objetos.

3.2.2. Dominio de Inteligencia (GPU Acelerada)

El módulo de predicción de atascos utiliza una red neuronal recurrente (LSTM). Dado que las operaciones matriciales de las redes neuronales son masivamente paralelas, este módulo se ejecuta sobre *TensorFlow* con soporte CUDA en la GPU. Esto permite una arquitectura asíncrona: mientras la CPU calcula la posición física de los coches en el siguiente paso de tiempo, la GPU procesa simultáneamente la probabilidad de bloqueo futuro, logrando una ejecución eficiente.

3.3. Validación experimental y análisis del proceso

Durante seis semanas consecutivas se ejecutaron múltiples pruebas experimentales de densidad, probabilidad, configuración de nodos y dinámica de flujos. Los principales hallazgos incluyen:

- El sistema presenta transiciones de fase libre–congestionada similares a las descritas por Boccara et al. (1993).
- Las intersecciones complejas generan atractores dinámicos estables.
- La autoorganización espontánea coincide con los modelos de Bar-Yam (1997).

Capítulo 4

RESULTADOS

4.1. Análisis del comportamiento del sistema y validación experimental

En esta sección se presentan los resultados obtenidos durante la experimentación del simulador de tráfico urbano **TeposIA**, desarrollado bajo un enfoque híbrido basado en autómatas celulares, modelación urbana discreta y computación de alto rendimiento. Los resultados permiten validar el comportamiento del sistema, evaluar la dinámica del flujo vehicular y analizar los fenómenos emergentes que se generaron a partir de la interacción entre vehículos, calles, intersecciones y probabilidades de giro.

El análisis se estructura en cuatro apartados: pruebas de densidad, evaluación de intersecciones, comportamiento emergente y métricas cuantitativas obtenidas durante la simulación.

4.2. Pruebas de densidad vehicular

El objetivo de estas pruebas fue observar cómo se comporta el sistema bajo diferentes niveles de carga vehicular. Se realizaron simulaciones en los niveles:

- **Densidad baja** ($\rho < 0{,}01$)
- **Densidad media** ($0{,}05 \leq \rho < 0{,}09$)
- **Densidad alta** ($0{,}15 \leq \rho < 0{,}25$)
- **Densidad crítica** ($\rho \geq 0{,}30$)

La densidad ρ se define como el cociente entre el número de celdas ocupadas y el número total de celdas disponibles por carril.

Resultados de las pruebas

Densidad baja. El sistema presenta un comportamiento estable, con velocidades promedio altas, congestión prácticamente inexistente y trayectorias limpias en

la mayoría de las calles. Los flujos obtenidos coinciden con las primeras aproximaciones de la Regla 184 descritas por Wolfram (2002), donde el tráfico se comporta de forma casi lineal.

Densidad media. Comienzan a aparecer fluctuaciones en el flujo vehicular. Se observa la formación de microcolas en intersecciones de alta demanda y un ligero incremento en tiempos de espera. Aun así, el sistema se mantiene en un régimen operativo.

Densidad alta. La dinámica cambia de manera significativa. Aparecen ondas de choque, cuellos de botella y un decrecimiento evidente en la velocidad promedio. Las intersecciones se vuelven zonas críticas y los giros comienzan a inducir perturbaciones locales en el flujo.

Densidad crítica. El sistema alcanza congestionamiento casi permanente. La velocidad promedio tiende a cero en múltiples nodos y se manifiestan patrones repetitivos, consistentes con los modelos descritos por Boccara, Fuks y otros autores sobre congestionamiento espontáneo en autómatas celulares de tráfico.

4.3. Evaluación del comportamiento en intersecciones

Dado que las intersecciones representan el punto más complejo del modelo urbano, se realizaron pruebas específicas para evaluar su comportamiento con diferentes probabilidades de giro y distintos niveles de carga vehicular.

Las probabilidades evaluadas incluyen:

- Giros desde Norte: recto, derecha, izquierda.
- Giros desde Sur: recto, derecha, izquierda.
- Giros desde Este: recto, derecha, izquierda.
- Giros desde Oeste: recto, derecha, izquierda.

Todas definidas en el JSON por nodo, lo cual permitió modificar el comportamiento de cada intersección de forma independiente, siguiendo una estructura modular.

Resultados en intersecciones

Los resultados más relevantes fueron:

- **Intersecciones equilibradas:** con probabilidades simétricas (0.7 recto, 0.15 giro), el tráfico fluyó correctamente y los bloqueos fueron mínimos.
- **Intersecciones con sesgo de giro:** aumentar la probabilidad de giro en uno o varios sentidos provocó congestión en carriles internos y modificaciones en la formación de colas.
- **Intersecciones con sobrecarga local:** nodos con alta demanda mostraron formación de ondas de choque y saturaciones regulares.

- **Intersecciones dependientes del mapa:** la estructura vial de Teposcolula posee avenidas más angostas, lo que incrementó los cuellos de botella.

Estos resultados coinciden con análisis de Bar-Yam (1997), donde pequeñas modificaciones en parámetros locales alteran el comportamiento global del sistema.

4.4. Fenómenos emergentes observados

Uno de los objetivos principales del proyecto fue analizar fenómenos emergentes en la simulación, entendidos como patrones macroscópicos que surgen a partir de reglas locales y microsituaciones en los nodos.

Los fenómenos más relevantes fueron:

- **Formación espontánea de colas:** una propiedad natural de los sistemas tipo Regla 184.
- **Ondas de choque:** formadas cuando un vehículo disminuye la velocidad en una región con alta densidad.
- **Congestión inducida:** situaciones donde un giro poco frecuente bloquea un carril completo.
- **Autoorganización:** en densidades específicas, el tráfico generó patrones repetitivos y ciclos estables.
- **Patrones periódicoss y atractores:** similares a los descritos por Wolfram en clases II y III.

En conjunto, estos fenómenos validan que el simulador cumple con las propiedades de un **sistema complejo discreto**, consistente con los modelos revisados en el marco teórico.

4.5. Métricas cuantitativas del sistema

Para cuantificar el comportamiento del tráfico se midieron los siguientes indicadores:

- **Flujo vehicular (q):** vehículos por unidad de tiempo.
- **Densidad (ρ):** relación entre celdas ocupadas y celdas totales.
- **Velocidad promedio (v):** promedio por carril/iteración.
- **Entropía de distribución:** medida del desorden.
- **Tiempo promedio de cruce:** desde punto inicial hasta el nodo final.

Los resultados generales fueron:

- Para densidad baja y media, el flujo q es alto y estable.

- Para densidad alta, el flujo decrece de forma no lineal.
- Para densidad crítica, el sistema presenta congestión total.
- La entropía aumenta en condiciones de tráfico mixto y disminuye en congestión.

Estas métricas sirven como base para sistemas de IA predictiva y control adaptativo.

4.6. Evaluación de Rendimiento Computacional

Dado que uno de los objetivos centrales del proyecto es la aplicación de Computación de Alto Rendimiento (HPC), se realizaron pruebas de estrés para cuantificar la eficiencia de la arquitectura híbrida implementada (Numba + Multiprocessing) frente a una implementación secuencial estándar en Python puro.

4.6.1. Impacto de la compilación JIT (Numba)

Se comparó el tiempo promedio de actualización por ciclo (*tick*) del autómata celular con y sin el decorador `@jit`. Las pruebas se realizaron con una densidad vehicular de $\rho = 0{,}3$ en una malla de 240×240 celdas.

Cuadro 4.1: Comparativa de tiempos de ejecución: Python Puro vs. Numba JIT

Operación	**Python (ms)**	**Numba JIT (ms)**	**Speedup (S)**
Cálculo de geometría (Bresenham)	0.82	0.04	20.5x
Actualización de matriz de estado	12.5	1.2	10.4x
Detección de colisiones	8.4	0.9	9.3x
Tiempo Total por Tick	**21.72 ms**	**2.14 ms**	**$\approx$ 10.1x**

Como se observa en la Tabla 4.1, la vectorización y compilación nativa permitieron una aceleración de un orden de magnitud, haciendo viable la simulación en tiempo real a 60 FPS incluso con alta carga vehicular.

4.6.2. Escalabilidad del Algoritmo Genético (Multiprocessing)

Para evaluar la eficiencia del módulo de optimización, se midió el tiempo requerido para completar una generación del Algoritmo Genético variando el número de procesos trabajadores (*workers*).

La eficiencia de paralelización E_p se mantuvo cercana al ideal lineal hasta los 8 núcleos, demostrando que la arquitectura de memoria compartida (SMP) minimiza el *overhead* de comunicación entre procesos.

- **Ejecución Serial (1 núcleo):** 180 segundos/generación.
- **Ejecución Paralela (4 núcleos):** 48 segundos/generación (Speedup $\approx$ 3.75x).

- **Ejecución Paralela (8 núcleos):** 26 segundos/generación (Speedup $\approx$ 6.9x).

Estos resultados validan que la arquitectura seleccionada aprovecha eficazmente los recursos de hardware disponibles, cumpliendo con los requerimientos no funcionales de alto rendimiento.

4.7. Validación del mapa urbano (JSON)

Se validó el correcto funcionamiento del mapa V7–V12 mediante:

- Comprobación de conectividad global.
- Revisión de orientación de carriles.
- Validación de probabilidades por nodo.
- Simulación de rutas largas.
- Pruebas de consistencia en zonas y segmentos.

Las validaciones confirmaron que el mapa está correctamente estructurado para simulaciones de tráfico urbano y para módulos de inteligencia artificial futuros.

4.8. Conclusiones generales de los resultados

Los resultados obtenidos permiten verificar que:

- El modelo reproduce comportamientos reales descritos por la literatura.
- La estructura del mapa urbano influye significativamente en el tráfico.
- Los parámetros locales alteran el comportamiento global del sistema.
- El simulador está listo para fases posteriores: IA, HPC y control adaptativo.

En conclusión, el sistema cumple con los objetivos planteados, reproduce fenómenos propios de los sistemas complejos, y constituye una plataforma válida para investigación, optimización y control inteligente del tráfico urbano.

Capítulo 5

CONCLUSIONES

El presente proyecto demostró que es posible modelar y analizar el tráfico urbano mediante el uso de autómatas celulares, estructuras discretas y computación de alto rendimiento. La construcción del simulador **TeposIA** permitió observar, de forma controlada y experimental, diversos fenómenos emergentes asociados al flujo vehicular, lo cual valida los fundamentos teóricos revisados y evidencia que los sistemas de tráfico constituyen una manifestación clara de los sistemas complejos descritos por Wolfram (2002), Bar-Yam (1997) y Adamatzky (2010).

A lo largo del desarrollo se identificaron tres contribuciones principales:

Contribución metodológica

La combinación de **Kanban** y **Prototipado Evolutivo** resultó ser una estrategia adecuada para un proyecto de naturaleza iterativa y basada en experimentación. Dicho enfoque permitió:

- integrar mejoras continuas a lo largo del desarrollo;
- mantener un flujo ordenado de tareas;
- probar rápidamente cada versión del simulador (V1–V12);
- corregir errores mediante iteraciones sucesivas;
- generar versiones funcionales desde etapas tempranas.

De esta forma, la metodología empleada no sólo guió el avance técnico, sino que además permitió validar hipótesis de comportamiento emergente en cada iteración.

Contribución técnica

El simulador TeposIA alcanzó un alto grado de estabilidad e integración técnica. Entre los logros más relevantes destacan:

- la construcción de un **motor de autómata celular extendido** capaz de manejar múltiples carriles, direcciones cardinales y probabilidades de giro;

- la representación del **mapa urbano de Teposcolula**, diseñado a partir de un archivo JSON estructurado y escalable;
- la integración de **intersecciones probabilísticas**, configurables por nodo y con soporte para comportamientos diferenciados;
- una **interfaz gráfica avanzada** con opciones de zoom, arrastre, control de velocidad y visualización del tráfico en tiempo real;
- un conjunto de **métricas cuantitativas** que permiten analizar el flujo, la densidad, la entropía del sistema y los patrones emergentes;
- una arquitectura preparada para módulos posteriores de **IA, AG e HPC**.

Estos avances consolidan una herramienta robusta para investigación, docencia y futuras implementaciones de control inteligente del tráfico urbano.

Contribución científica

Los resultados experimentales obtenidos permitieron identificar fenómenos característicos de los sistemas complejos, como:

- formación espontánea de colas;
- ondas de choque vehicular;
- ciclos y atractores de tráfico;
- congestión inducida por condiciones locales;
- autoorganización bajo parámetros específicos.

Estos fenómenos coinciden con modelos teóricos reportados en la literatura, indicando que el simulador reproduce adecuadamente dinámicas reales del tráfico. Esto reafirma que los autómatas celulares son un modelo válido para el estudio del transporte urbano, especialmente cuando se busca capturar efectos emergentes sin necesidad de ecuaciones continuas ni modelos diferenciales complejos.

Conclusión general

En conjunto, el proyecto alcanzó los objetivos planteados: se construyó un simulador urbano capaz de modelar de forma realista el comportamiento del tráfico de Teposcolula, se validaron fenómenos emergentes propios de sistemas complejos, y se generó una plataforma estable para futuras líneas de investigación.

Además, se estableció una base sólida para integrar modelos de **inteligencia artificial**, **algoritmos evolutivos** y **computación de alto rendimiento**, abriendo la posibilidad de crear un sistema de control inteligente del tráfico adaptable a condiciones reales.

El simulador TeposIA constituye un avance significativo en la comprensión, análisis y gestión de dinámicas urbanas complejas, y representa una herramienta funcional para el estudio y mejora del flujo vehicular en ciudades pequeñas y medianas.

Capítulo 6

COMPETENCIAS DESARROLLADAS

6.1. Competencias desarrolladas y/o aplicadas

Durante el desarrollo del proyecto **Control de sistemas complejos a través de medios no lineales con computación de alto rendimiento**, se fortalecieron y aplicaron diversas competencias profesionales, técnicas y transversales propias del perfil del Ingeniero en Sistemas Computacionales. Estas competencias se desarrollaron mediante la implementación del simulador urbano TeposIA, el análisis de modelos computacionales y la experimentación con sistemas complejos discreto-dinámicos.

A continuación se describen las principales competencias desarrolladas y/o aplicadas:

Competencias técnicas y de ingeniería

- **Modelación computacional:** Aplicación de autómatas celulares, modelos discretos y principios de sistemas complejos para representar dinámicas urbanas de tráfico, siguiendo marcos teóricos como Wolfram, Bar-Yam y Adamatzky.

- **Desarrollo de software avanzado:** Implementación de un simulador modular en Python, integrando estructuras de datos, procesamiento de información, manejo de archivos JSON, concurrencia de procesos y diseño escalable.

- **Diseño e implementación de estructuras de datos:** Creación y uso de estructuras para representar calles, carriles, nodos, probabilidades, flujos, zonas y topología urbana a nivel discreto.

- **Simulación y análisis de sistemas:** Ejecución y evaluación de múltiples escenarios de simulación para observar el comportamiento emergente, fenómenos de congestión, ondas de choque y autoorganización.

- **Optimización computacional:** Preparación del motor de simulación para futuras mejoras con paralelización, GPU y herramientas HPC.

Competencias metodológicas

- **Aplicación de metodologías ágiles:** Uso del enfoque Kanban para la organización del flujo de trabajo, priorización de tareas y control del avance del proyecto.
- **Prototipado evolutivo:** Desarrollo incremental de versiones funcionales (V1–V12), validación y mejora continua basada en experimentación.
- **Gestión de versiones y control de cambios:** Documentación ordenada del progreso, registro de ajustes y evolución del sistema.
- **Diseño iterativo:** Capacidad para analizar, rediseñar y mejorar componentes del sistema con base en resultados experimentales.

Competencias de investigación y análisis

- **Integración de fundamentos teóricos:** Comprensión y aplicación de teorías de sistemas complejos, computación no lineal, automatización celular y dinámica urbana.
- **Análisis de fenómenos emergentes:** Identificación y estudio de patrones de tráfico, congestión y comportamientos globales derivados de reglas locales.
- **Validación de modelos:** Comparación de resultados obtenidos con la literatura especializada y modelos teóricos reportados.
- **Capacidad crítica y de evaluación:** Identificación de limitaciones del modelo, análisis de errores y formulación de propuestas de mejora.

Competencias tecnológicas

- **Programación avanzada en Python:** Uso de librerías para simulación, estructuras de datos, interfaces gráficas y manejo de archivos.
- **Diseño y estructuración de archivos JSON:** Creación de un formato escalable y estandarizado para representar mapas urbanos de forma modular.
- **Desarrollo de interfaces gráficas:** Implementación de una UI interactiva con zoom, arrastre, ejecución paso a paso y visualización en tiempo real.
- **Conceptos de HPC (High Performance Computing):** Preparación del código para optimizaciones paralelas y uso futuro de procesamiento en GPU.

Competencias transversales y profesionales

- **Pensamiento crítico y resolución de problemas:** Abordaje de problemas complejos como congestión, inconsistencias en nodos, gridlocks y patrones no previstos.
- **Comunicación técnica:** Documentación clara del desarrollo, resultados, conclusiones y estructura del proyecto.

- **Trabajo autónomo y gestión del tiempo:** Administración del progreso semana a semana, cumplimiento de objetivos y autoorganización en el desarrollo.
- **Aprendizaje continuo:** Integración de nuevos modelos, técnicas y herramientas computacionales conforme avanzó el proyecto.
- **Responsabilidad profesional:** Compromiso con la calidad, rigurosidad científica y correcto manejo de información técnica.

Bibliografía

Adamatzky, A. (2010). *Game of Life Cellular Automata*. Springer, London.

Adamatzky, A. and Martínez, G. J. (2016). *Designing Beauty: The Art of Cellular Automata*. Springer.

Adamatzky, A., Martínez, G. J., and Seck-Tuoh-Mora, J. C. (2006). Phenomenology of reaction-diffusion binary-state cellular automata. *International Journal of Bifurcation and Chaos*, 16(10):2985–3005.

Bar-Yam, Y. (1997). *Dynamics of Complex Systems*. Perseus Books, Reading, MA.

Boccara, N. and Fuks, H. (1993). Traffic flow on a ring with a bottleneck. *Journal of Physics A: Mathematical and General*, 26(15):3701.

Chowdhury, D., Santen, L., and Schadschneider, A. (2000). Statistical physics of vehicular traffic and some related systems. *Physics Reports*, 329(4-6):199–329.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation*. Prentice Hall.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Lam, S. K., Pitrou, A., and Seibert, S. (2023). Numba: A high performance python compiler. `https://numba.pydata.org/`. Accedido: 2025.

Maerivoet, S. and De Moor, B. (2005). Transportation planning and traffic engineering: A cellular automata approach. *Transportation Research Part C: Emerging Technologies*, 13(4):299–325.

Magnier, M., Lattaud, C., and Heudin, J.-C. (1997). Complexity classes in the two-dimensional life cellular automata subspace. *Complex Systems*, 11(6):419–436.

Mainzer, K. and Chua, L. O. (2011). *Local Activity Principle: The Cause of Complexity and Symmetry Breaking*. Imperial College Press.

Martínez, G. J. (2013). A note on elementary cellular automata classification. *Journal of Cellular Automata*, 8(3-4):233–259. arXiv preprint arXiv:1306.5577.

Morita, K. (2012). Computation in reversible cellular automata. *International Journal of General Systems*, 41(6):563–581.

Morita, K. (2017). *Theory of Reversible Computing*. Springer, Japan.

Nagel, K. and Schreckenberg, M. (1992). A cellular automaton model for freeway traffic. *Journal of Physics I*, 2(12):2221–2229.

Packard, N. H. and Wolfram, S. (1985). Two-dimensional cellular automata. *Journal of Statistical Physics*, 38(5):901–946.

Wolfram, S. (1983). Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55(3):601–644.

Wolfram, S. (2002). *A New Kind of Science*. Wolfram Media, Champaign, IL.

Zhang, H., Feng, S., Liu, C., Ding, Y., et al. (2019). Cityflow: A multi-agent reinforcement learning environment for large scale city traffic scenario. In *The World Wide Web Conference*, pages 3620–3624.

ANEXOS

A continuacion, se muestra el link hacia el repositorio de GitHub donde se encuentra el codigo y el ejecutable de este proyecto.
Link del repositorio en GitHub

6.2. Manual de Usuario del Sistema TeposIA

Este manual describe la instalación, configuración y operación del simulador de tráfico urbano **TeposIA** (Versión HPC/IA). El sistema se divide en cuatro módulos principales: Editor de Mapas, Simulador de Tráfico, Optimizador Evolutivo y Predictor Neuronal.

6.2.1. 1. Requisitos e Instalación

Requisitos del Sistema

- **Sistema Operativo:** Windows 10/11, Linux (Ubuntu 20.04+) o macOS.
- **Procesador:** Recomendado 4 núcleos o más (para el Optimizador Genético).
- **GPU:** Opcional pero recomendada (NVIDIA RTX) para la aceleración del módulo de predicción.
- **Python:** Versión 3.10 o superior.

Instalación de Dependencias

Antes de ejecutar el simulador, es necesario instalar las librerías científicas y gráficas. Ejecute el siguiente comando en la terminal:

```
pip install numpy matplotlib pillow tensorflow scikit-learn joblib numba sv-ttk
```

6.2.2. 2. Módulo 1: Editor de Topología (Editor)

Al iniciar la aplicación, se presenta la pestaña **Editor**. Esta herramienta permite diseñar o modificar la red vial de Teposcolula.

Herramientas de Diseño

- **Normal (N):** Modo de navegación. Permite arrastrar el mapa y hacer zoom.

- **Calle:** Haga clic izquierdo para definir los puntos de una nueva calle. Al terminar, suelte el clic y confirme. Se solicitará el ID, nombre, número de carriles (1-3) y sentido de circulación.
- **Zona:** Permite dibujar polígonos decorativos (parques, edificios) para dar contexto visual.
- **Bloquear (B):** Herramienta crítica. Haga clic en cualquier celda de la calle para crear un bloqueo permanente (simulando obras o accidentes estáticos).

Gestión de Elementos

- **Panel de Calles:** Seleccione una calle de la lista para editar sus propiedades:
  - *Cerrar/Abrir:* Habilita o deshabilita el flujo en esa vía.
  - *Prioridad:* Defina si es "Principal"(preferencia de paso) o "Secundaria".
- **Panel de Nodos:** Permite editar las probabilidades de giro.
  - *Formato Cardinal:* Defina la probabilidad de ir al Norte, Sur, Este u Oeste.
  - *Formato Origen (AG):* Define probabilidades relativas (Recto, Derecha, Izquierda) según de dónde viene el vehículo.

6.2.3. 3. Módulo 2: Simulador de Tráfico

Es el núcleo del sistema donde se visualiza la dinámica vehicular.

Controles de Ejecución

- **Iniciar/Pausar:** Controla el ciclo de simulación. (Atajo: Barra Espaciadora).
- **Reiniciar:** Restablece el estado, limpia vehículos y métricas, y reinicia el reloj a las 07:00 AM del Lunes.
- **Exportar CSV:** Guarda el historial de flujo, densidad y entropía en un archivo para análisis externo.

Configuración Dinámica

- **Perfil de Tiempo:** El sistema alterna automáticamente entre perfiles "Díaz "Noche"según el reloj simulado, afectando el volumen de tráfico y tipos de vehículos (más camiones de noche).
- **Generación (Spawn):**
  - *Mixto:* Genera vehículos tanto en las entradas del mapa como en posiciones aleatorias internas.
  - *Flujo (Slider):* Multiplicador de la cantidad de vehículos generados.
- **Física:**

- *Habilitar Accidentes:* Si está activo, los vehículos detenidos por mucho tiempo pueden sufrir averías, bloqueando el carril hasta que el sistema los elimine (probabilidad de grúa).

Interacción en Tiempo Real

El usuario puede alterar la simulación mientras corre:

- **Poner Coche:** Seleccione el tipo (Auto, Moto, Camión) y haga clic en un carril para inyectar un vehículo manualmente.
- **Bloqueo Dinámico:** Haga clic en una celda para crear un obstáculo temporal.

6.2.4. 4. Módulo 3: Optimizador (Algoritmo Genético)

Utiliza Computación de Alto Rendimiento (HPC) para encontrar la configuración ideal de semáforos.

Pasos para optimizar:

1. Configure los parámetros evolutivos:
   - *Generaciones:* Cantidad de ciclos evolutivos (Recomendado: 50+).
   - *Población:* Número de soluciones candidatas por generación (Recomendado: 20-40, depende de los núcleos de su CPU).
2. Presione **Ïniciar Optimización"**.
3. El sistema utilizará *Multiprocessing* para evaluar múltiples simulaciones en paralelo. La barra de progreso mostrará el avance.
4. Al finalizar, se presentará la mejor solución encontrada (mayor flujo, menor entropía) y se ofrecerá la opción de guardar un nuevo mapa JSON optimizado.

6.2.5. 5. Módulo 4: Predictor (Inteligencia Artificial)

Este módulo utiliza una Red Neuronal LSTM para predecir atascos antes de que ocurran.

Flujo de Trabajo:

1. **Recolección de Datos:** En la pestaña Predictor, configure el número de simulaciones y presione Ïniciar Recolección". Esto generará el archivo `predictor_data.csv` con miles de muestras de tráfico.
2. **Entrenamiento:** Ejecute el script externo `entrenador.py`. Este script leerá el CSV, entrenará la red neuronal usando TensorFlow/GPU y generará los archivos `predictor_modelo.h5` y `scaler.joblib`.
3. **Inferencia:** En el simulador, presione Cargar Modelo. Si los archivos existen, el sistema comenzará a mostrar predicciones en tiempo real en la esquina superior (ej. "PREDICCIÓN: ATASCO INMINENTE").

6.3. Glosario

Algoritmo Genético (AG): Técnica de búsqueda y optimización inspirada en el proceso de selección natural, utilizada en este proyecto para evolucionar la configuración de los semáforos.

Autómata Celular (AC): Modelo matemático discreto compuesto por una rejilla de celdas que cambian de estado según reglas locales, usado para simular el flujo vehicular.

Batch Size: Número de muestras de entrenamiento que se propagan a través de la red neuronal antes de actualizar los pesos internos del modelo.

Bresenham (Algoritmo): Algoritmo eficiente para trazar líneas en gráficos rasterizados, utilizado en el simulador para digitalizar las coordenadas de las calles sobre la malla discreta.

Caminata Aleatoria (Random Walk): Objeto matemático que describe una trayectoria que consiste en una sucesión de pasos aleatorios, aplicado al movimiento estocástico de los agentes en la red vial.

Compilación JIT (Just-In-Time): Técnica que transforma código interpretado (como Python) a código máquina nativo durante la ejecución para mejorar el rendimiento.

CUDA: (Compute Unified Device Architecture) Plataforma de computación paralela y modelo de programación de NVIDIA que permite usar la GPU para propósitos generales.

Dropout: Técnica de regularización en redes neuronales donde se ignoran aleatoriamente neuronas durante el entrenamiento para prevenir el sobreajuste.

Emergencia: Propiedad de los sistemas complejos donde patrones y comportamientos globales surgen de la interacción de componentes simples, sin un control centralizado.

Entropía de Shannon: Medida de la incertidumbre o desorden en un sistema de información, utilizada aquí para cuantificar la distribución espacial de los vehículos.

Epoch (Época): Un ciclo completo a través de todo el conjunto de datos de entrenamiento durante el aprendizaje de una red neuronal artificial.

Fitness (Aptitud): Valor numérico que representa la calidad de una solución en un algoritmo genético; en este caso, una combinación de flujo vehicular y entropía.

GIL: (Global Interpreter Lock) Mecanismo de Python que impide que múltiples hilos nativos ejecuten bytecodes simultáneamente, superado mediante *Multiprocessing*.

Gridlock: Estado de congestión total en una red de tráfico donde el movimiento se detiene debido a bloqueos cíclicos en las intersecciones.

HPC: (High Performance Computing) Uso de superordenadores o técnicas de procesamiento paralelo para resolver problemas computacionales complejos en menor tiempo.

JSON: (JavaScript Object Notation) Formato ligero de intercambio de datos, utilizado en el proyecto para almacenar la topología del mapa urbano y configuraciones.

Kanban: Metodología ágil de gestión de proyectos que utiliza tableros visuales para controlar el flujo de trabajo y limitar las tareas en progreso.

LSTM: (Long Short-Term Memory) Tipo de red neuronal recurrente capaz de aprender dependencias a largo plazo, ideal para predecir series temporales como el tráfico.

Multiprocessing: Uso de dos o más unidades centrales de procesamiento (CPU) dentro de un solo sistema informático, permitiendo la ejecución paralela real de procesos.

No linealidad: Característica de sistemas donde la salida no es directamente proporcional a la entrada, fundamental en la dinámica del tráfico urbano.

Numba: Compilador de código abierto que traduce un subconjunto de Python y NumPy a código máquina rápido, optimizando las funciones críticas del simulador.

NumPy: Biblioteca fundamental para la computación científica en Python que proporciona soporte para arreglos multidimensionales y matrices de alto rendimiento.

Prototipado Evolutivo: Metodología de desarrollo de software donde el sistema se construye mediante versiones sucesivas que incorporan mejoras incrementales.

Regla 184: Autómata celular elemental unidimensional utilizado comúnmente para modelar el flujo de tráfico libre de colisiones.

SIMD: (Single Instruction, Multiple Data) Técnica de paralelismo en la que una sola instrucción opera simultáneamente sobre múltiples datos, aprovechada por las operaciones vectorizadas.

Sistema Complejo: Sistema compuesto por muchas partes interconectadas que, como un todo, exhiben propiedades que no son evidentes a partir de la suma de las partes individuales.

SMP: (Symmetric Multiprocessing) Arquitectura de hardware donde múltiples procesadores idénticos están conectados a una única memoria principal compartida.

Speedup: Métrica que cuantifica la mejora en la velocidad de ejecución de una tarea al ser paralelizada en comparación con su ejecución secuencial.

TensorFlow: Biblioteca de código abierto para aprendizaje automático desarrollada por Google, utilizada para construir y entrenar el modelo predictor de tráfico.

Vectorización: Proceso de convertir un algoritmo que opera con un solo valor a la vez en uno que opera con un conjunto de valores (vectores) simultáneamente.