Tuyệt vời — đề tài của bạn rất “đắt” vì SAGSINs (Space–Air–Ground–Sea Integrated Networks) đúng là môi trường đa tầng, đa miền, cực động. Dưới đây là lộ trình thực hiện từ A→Z để bạn có thể triển khai ngay, theo hướng dùng GenAI như “bộ não hợp tác” giúp giảm độ trễ.

**1) Mục tiêu & phạm vi (rõ ràng, đo được)**

* **Mục tiêu chính**: Giảm **độ trễ đầu-cuối (E2E latency)** trong SAGSINs bằng cơ chế **hợp tác giữa vệ tinh–UAV–trạm mặt đất–tàu biển** do GenAI gợi ý/điều phối.
* **Mục tiêu phụ**:
  + Tăng **tỉ lệ chuyển phát gói** (PDR), giảm **jitter**, cân bằng tải, sử dụng tài nguyên hiệu quả.
  + Phản ứng tốt trước **nhu cầu/điều kiện kênh biến động** (ví dụ nhiễu, tắc nghẽn, thời tiết, quỹ đạo).
* **Kịch bản tối thiểu**: Constellation LEO + đàn UAV + các BS mặt đất + 1–2 tuyến hàng hải; lưu lượng đa dịch vụ (video/IoT/mission-critical).

**2) Ý tưởng cốt lõi (GenAI đóng vai gì?)**

GenAI không chỉ “dự đoán” mà còn “đề xuất chiến lược hợp tác”:

1. **World Model (Mô hình thế giới sinh/tiên đoán)**
   * Mô hình chuỗi thời gian (Transformer hoặc latent diffusion cho chuỗi) dự đoán **trạng thái nút/đường link** ở bước tới: băng thông khả dụng, SNR, hàng đợi, vị trí UAV, ma trận lưu lượng.
   * Đầu ra giúp **trước-khi-xảy-ra** (proactive) chọn tuyến, lịch truyền, tái cấu hình.
2. **Policy Generator (Sinh chính sách hợp tác)**
   * Một **LLM/GNN-Transformer** trên đồ thị mạng sinh ra **kế hoạch định tuyến + lập lịch + cấp phát tài nguyên** dạng token/chuỗi (ví dụ “UAV\_3 làm relay cho tàu T1 trong 10s; uplink qua Sat-12; ưu tiên luồng video #5”), rồi **tinh chỉnh** bằng MARL.
   * Có thể dùng **chain-of-thought nội bộ** để tự giải thích ràng buộc (độ trễ, công suất, quỹ đạo), sau đó rút gọn thành **hành động hợp lệ**.
3. **Scenario/Traffic Generator (Sinh kịch bản & dữ liệu)**
   * Mô hình sinh (VAE/diffusion) tạo **topology/traffic synthetic** đa dạng để huấn luyện/kiểm thử (giúp bao phủ nhiều tình huống hiếm).

Kết hợp 3 khối trên → hệ thống **dự đoán–lập kế hoạch–thực thi** theo vòng lặp khép kín.

**3) Kiến trúc hệ thống đề xuất**

**Tầng sensing & trạng thái**

* Telemetry từ Sat/UAV/BS/Tàu: {SNR, BER, queue, vị trí, tốc độ, năng lượng, dự báo thời tiết/biển}.

**Tầng GenAI**

* **World Model** (Transformer/Latent State-Space): dự đoán trạng thái t+1..t+H.
* **Policy Generator** (Graph Transformer + LLM): sinh hành động (route/schedule/power).
* **MARL Finetuning** (PPO/APPO/QMIX/MAPPO): tối ưu đa mục tiêu với feedback thực.

**Tầng điều phối phân tán**

* **Contract Net / Auction** nhẹ hoặc **federated coordination**: các nút thương lượng slot/beam/time-window.
* **Safety Shield**: kiểm tra ràng buộc (băng thông, công suất, QoS) trước khi áp dụng.

**Tầng thực thi**

* SDN controller (trên mặt đất + gateway vệ tinh) đẩy rule/flow; UAV nhận plan qua control channel.

**4) Thiết lập mô phỏng & dữ liệu**

* **Trình giả lập mạng**:
  + **ns-3** (+ ns3-gym) hoặc **OMNeT++** (INET) để đo latency/PDR chính xác.
  + **NetworkX** cho tiền xử lý đồ thị & script nhanh.
* **Quỹ đạo**: TLE/SGP4 giả lập LEO; UAV theo waypoint/boids; tàu theo tuyến hàng hải.
* **Lưu lượng**: Poisson/Bursty (ON/OFF), video (CBR/VBR), IoT small packets.
* **Dữ liệu huấn luyện**:
  + Thu từ mô phỏng rollouts (state, action, reward, next\_state).
  + Tạo thêm bằng **data augmentation** từ generator.

**5) Bài toán tối ưu & phần thưởng (reward)**

* **Mục tiêu**: Minimize E[E2E latency]\mathbb{E}[\text{E2E latency}] + penalty cho jitter, drop, năng lượng.
* **Reward ví dụ (mỗi bước)**:

rt=−α⋅L‾t−β⋅Jittert−γ⋅DropRatet−η⋅Energyt+λ⋅Fairnesstr\_t = - \alpha \cdot \overline{L}\_t - \beta \cdot \text{Jitter}\_t - \gamma \cdot \text{DropRate}\_t - \eta \cdot \text{Energy}\_t + \lambda \cdot \text{Fairness}\_t

* **Ràng buộc**: Delay ≤ Dmax, BER ≤ Bmax, năng lượng UAV ≤ Ebudget, link capacity.
* **Multi-objective**: Dùng scalarization (trọng số) hoặc Pareto RL.

**6) Baseline để so sánh (rất quan trọng)**

* **Định tuyến cổ điển**: Dijkstra/OSPF-like trên latency ước lượng.
* **DTN/Contact-Graph** cho vệ tinh (CGRA).
* **Heuristic hợp tác**: “closest-UAV relay”, “max-SNR link”, “round-robin scheduling”.
* **RL không GenAI**: MARL trực tiếp không dùng world model hay policy generator.

**7) Thử nghiệm & chỉ số đánh giá**

* **Latency (p50/p95/p99)**, **Jitter**, **PDR**, **Throughput**, **Fairness (Jain’s index)**, **Utilization**, **Energy/UAV flight time**.
* **Khả năng khái quát**: chạy kịch bản chưa thấy trước (khác quỹ đạo/nhu cầu).
* **Khả năng phục hồi**: link lỗi, thời tiết xấu, UAV rớt pin, tắc nghẽn đột biến.

**8) Quy trình thuật toán (vòng lặp phối hợp)**

1. Thu thập trạng thái sts\_t từ các nút (đồ thị động + đặc trưng link/queue).
2. **World Model** dự đoán s^t+1:t+H\hat{s}\_{t+1:t+H}.
3. **Policy Generator** sinh kế hoạch ata\_t (route/schedule/power/relay assignment).
4. **Shield** kiểm ràng buộc → at′a\_t'.
5. Thực thi at′a\_t' trong mô phỏng → quan sát st+1s\_{t+1}, nhận reward rtr\_t.
6. **MARL finetune** để cập nhật policy; **generator** cũng học từ phản hồi.
7. Định kỳ sinh **kịch bản mới** (generator) để cải thiện độ bền.

**Pseudocode rút gọn (không phụ thuộc công cụ)**:

Init world\_model, policy\_generator, MARL\_policy

for episode in scenarios:

G = init\_dynamic\_graph() # Sat-Air-Ground-Sea topology

s = observe\_state(G)

for t in 1..T:

s\_pred = world\_model.predict(s, horizon=H)

plan = policy\_generator.generate(G, s, s\_pred, goals={min\_latency,...})

a = shield(plan, constraints)

s2, r = simulate\_step(G, a)

MARL\_policy.update(s, a, r, s2)

policy\_generator.refine\_with\_feedback(s, a, r, s2)

s = s2

Evaluate against baselines on held-out scenarios

**9) Công cụ & thư viện gợi ý**

* **Mô phỏng & RL**: Python, **ns-3 + ns3-gym**, **Ray RLlib** / CleanRL / Stable-Baselines3, **Gymnasium**.
* **Học sâu**: **PyTorch**, PyTorch Geometric (GNN).
* **Quỹ đạo**: sgp4 (TLE), poliastro (ngoại tuyến).
* **Quản lý thí nghiệm**: Weights & Biases / MLflow, Hydra.
* **Đồ thị**: NetworkX, torch\_geometric.

**10) Kế hoạch 12 tuần (thực chiến)**

**Tuần 1–2**:

* Tổng hợp yêu cầu, thiết kế KPI; dựng **ns-3** topology tối thiểu (LEO + UAV + BS + tàu).
* Log telemetry (latency, queue, SNR). Viết data pipeline.

**Tuần 3–4**:

* Huấn luyện **World Model** dự đoán băng thông/queue/latency short-term.
* Xây baseline định tuyến cổ điển + heuristic relay.

**Tuần 5–6**:

* Xây **Policy Generator** (Graph Transformer hoặc LLM-lập-kế-hoạch cho đồ thị).
* Tích hợp **shield** kiểm ràng buộc.

**Tuần 7–8**:

* **MARL finetuning** (MAPPO/APPO) với reward đa mục tiêu.
* Khép vòng lặp dự đoán–sinh–thực thi.

**Tuần 9**:

* **Scenario Generator** (VAE/diffusion) tạo topology/traffic mới; kiểm tra khả năng khái quát.

**Tuần 10–11**:

* So sánh với baselines trên bộ kịch bản chuẩn; phân tích độ nhạy (độ nhiễu, mất link).
* Tối ưu tốc độ suy luận (bộ nhớ, batching), kiểm tra ổn định.

**Tuần 12**:

* Viết báo cáo, biểu đồ, ablation study: (w/wo world model, w/wo MARL, w/wo generator).
* Chuẩn bị demo: 2–3 kịch bản minh họa giảm p95 latency.

**11) Thiết kế dữ liệu đầu vào mô hình (gợi ý đặc trưng)**

* **Node features**: loại nút (sat/UAV/BS/ship), vị trí 3D, vận tốc, năng lượng (UAV), queue len, tx power.
* **Edge features**: SNR/BER, băng thông danh định & khả dụng, RTT, nhiễu, khoảng cách, tần số/beam ID.
* **Traffic features**: class (video/IoT/URLLC), rate, deadline.
* **Context**: thời tiết/biển, lịch quỹ đạo (contact plan), vùng cấm bay.

**12) Phương pháp học chi tiết**

* **World Model**:
  + Seq2Seq Transformer / Temporal GNN (TGAT/TGN) / State-Space (S4/SGRU) cho đồ thị động.
  + Loss: MSE cho thông số liên tục + CRPS nếu muốn dự báo phân phối.
* **Policy Generator**:
  + **Graph Transformer** sinh chuỗi hành động (routing path + slot assignment).
  + Huấn luyện **supervised** từ nghiệm heuristic tốt + **reinforcement** qua reward môi trường.
  + Có thể dùng **IL + RL** (behavior cloning → RL fine-tune).
* **MARL**:
  + **MAPPO/APPO** cho các agent (sat cluster, UAV, BS).
  + **Centralized training, decentralized execution** (CTDE).

**13) Rủi ro & cách kiểm soát**

* **Không ổn định khi sinh chiến lược** → thêm **safety shield** + penalty mạnh cho vi phạm.
* **Chi phí suy luận cao** → lượng tử hóa/ONNX/tiny-TensorRT; giảm tần suất tái lập kế hoạch.
* **Generalization kém** → đa dạng kịch bản (generator), domain randomization.
* **Thiếu dữ liệu thực** → mô phỏng phong phú + (nếu được) **federated learning** tích lũy trên hiện trường.

**14) Đầu ra học thuật & kỹ thuật**

* **Bộ mã mô phỏng + huấn luyện** (ns-3 + PyTorch/RLlib).
* **Benchmark kịch bản công khai** (cấu hình, hạt giống).
* **Số liệu**: cải thiện p95/p99 latency, PDR, fairness, năng lượng.
* **Ablation**: vai trò từng khối (world model, generator, MARL).
* **Bài báo/luận văn**: mô tả kiến trúc, thuật toán, chứng minh thực nghiệm.

**15) Khung API tối thiểu (gợi ý để bạn code)**

# pseudo-interfaces (Pythonic)

class SAGEnv: # ns3-gym wrapper

def reset(self, scenario\_cfg) -> dict: ...

def step(self, action\_dict) -> tuple[dict, float, bool, dict]: ...

def observe(self) -> dict: ... # node/edge features

class WorldModel(torch.nn.Module):

def forward(self, state\_graph, horizon:int=5) -> list[PredState]: ...

class PolicyGenerator(torch.nn.Module):

def forward(self, state\_graph, pred\_graph\_seq, goals) -> Plan: ...

def decode\_plan(self, plan) -> ActionDict: ...

class SafetyShield:

def enforce(self, plan, constraints) -> ActionDict: ...

class MARLPolicy: # e.g., MAPPO with Ray RLlib

def act(self, obs) -> ActionDict: ...

def update(self, s, a, r, s2): ...

**16) Bạn làm gì ngay hôm nay?**

1. Cài **ns-3 + ns3-gym** và dựng **topology tối thiểu** (1 plane LEO, 3 UAV, 2 BS, 1 tàu).
2. Log 6 đặc trưng: {RTT, SNR, queue, throughput, loss, energy}.
3. Huấn luyện **World Model** dự đoán RTT/throughput 5 bước tới.
4. Dùng **Graph shortest path** + **slot scheduling đơn giản** làm baseline.
5. Thêm **Policy Generator** (Graph Transformer nhỏ) sinh route+slot, áp dụng shield, đo latency p95 → so với baseline.

Nếu bạn muốn, mình có thể:

* Tạo skeleton code (Python/PyTorch + ns3-gym)
* Viết file cấu hình kịch bản ns-3 mẫu
* Phác thảo kiến trúc Graph Transformer cho Policy Generator

Chỉ cần nói bạn chọn **ns-3** hay **OMNeT++**, và bạn muốn mình **đưa skeleton** theo stack nào, mình sẽ thả ngay bộ khởi động để bạn chạy được luôn.