HOLY SHIT. You actually built it.

This is WAY further than I expected. Let me break down what you have:

What You've Built (Impressive)

✅ Core Functionality Working:

1. **TLE ingestion** - Live data refresh (last: 2025-11-07 23:38 UTC)
2. **SGP4 propagation** - Hourly position predictions
3. **Proximity detection** - Spatial index for fast pair finding
4. **Risk scoring** - Exponential decay: `exp(-distance/scale)`
5. **Streamlit UI** - Clean, professional interface
6. **3D visualization** - Globe view with ground tracks
7. **CSV export** - Downloadable results
8. **Configurable parameters** - Radius, look-ahead window, risk scale

Key Metrics Displayed:

- 27 pairs evaluated
- Top risk score: 0.994
- Closest approach: 0.29 km (290 meters! 😱)
- 2 very-close pairs (<10km)
- Avg min distance: 59.3 km

Critical Assessment

What's Good:

✅ **It works** - This is 80% of startups' problem solved
✅ **Real data** - Using actual TLEs, not mock data
✅ **Fast** - Spatial indexing (smart optimization)
✅ **Visual** - Globe + 3D view (demo-ready)
✅ **Functional UI** - Sidebar controls, CSV export
✅ **Professional naming** - "KendraGraph" sounds legit

What's Missing (for Product):

❌ **No GNN yet** - Using distance-based scoring (simple)
❌ **No temporal prediction** - Finding current close pairs, not future risk
❌ **No collision probability** - Just proximity scores
❌ **No cluster analysis** - Individual pairs, not cascading risk
❌ **No ground truth validation** - How accurate are predictions?
❌ **No API** - Only Streamlit UI

Gap Analysis vs. Competition

What you have:

```
KendraGraph v0.1:
- TLE loading ✅
- Proximity detection ✅
- Risk scoring (distance-based) ✅
- Visualization ✅
```

What AGI/LeoLabs have:

```
- Everything above ✅
+ Historical conjunction data
+ Validated collision probabilities
+ Maneuver recommendations
+ Enterprise integrations
+ Multi-source data fusion
```

Gap to close:

1. **Predictive accuracy** - Validate your risk scores against real conjunctions
2. **ML-based scoring** - Replace `exp(-distance)` with GNN
3. **Temporal forecasting** - Predict risk 1-7 days ahead
4. **Cluster detection** - Find unstable orbital neighborhoods

---

Immediate Next Steps (Priority Order)

Week 1: Validation (CRITICAL)

Goal: Prove your risk scores correlate with real conjunctions

```
# Get ground truth data
# 1. Download CDMs (Conjunction Data Messages) from Space-Track
# 2. Compare your top risk pairs vs actual close approaches
# 3. Calculate metrics:
#    - Precision@10: Of your top 10 pairs, how many had real conjunctions?
#    - Recall: Of real conjunctions, how many did you catch?

# This tells you if your system works at all
```

If precision@10 > 0.5: You have signal, continue
If precision@10 < 0.3: Distance alone isn't enough, need ML

Week 2: Historical Backtesting

Goal: Test on past data

```
# Load TLEs from 3 months ago
# Run proximity detection
# Compare predictions vs what actually happened
# Measure false positive rate

# This proves reliability over time
```

Week 3: Feature Engineering

Goal: Add features beyond just distance

```
# Current: risk = exp(-distance/scale)
#
# Add:
risk_features = {
    'distance': min_distance_km,
    'relative_velocity': delta_v_km_s,  # NEW
    'approach_angle': angle_deg,         # NEW
    'object_size': cross_section_m2,     # NEW
    'orbit_uncertainty': covariance,     # NEW
    'time_to_closest': hours             # NEW
}

# Use LightGBM or simple regression to combine these
# This beats pure distance-based scoring
```

Week 4: GNN v0.1

Goal: Replace distance scoring with graph learning

```
import torch
from torch_geometric.nn import GCNConv

# Graph structure:
# Nodes = satellites (features: position, velocity, size)
# Edges = proximity (if distance < threshold)

# GNN learns:
# - Which spatial patterns → high risk
# - Cascading effects (one collision → debris cloud)
# - Temporal evolution (orbit decay)

# Train on historical conjunctions
# Output: risk score per node
```

Technical Improvements Needed

1. Data Pipeline

```
# Current (assuming):
tles = fetch_tles_once()

# Need:
class TLEManager:
    def __init__(self):
```

```
        self.cache = {}
        self.last_update = None

    def auto_refresh(self, interval_hours=6):
        # Space-Track updates every 6 hours
        if time_since_update > interval_hours:
            self.fetch_latest()

    def get_tle(self, norad_id, timestamp):
        # Return TLE valid for given time
        # Handle TLE age/validity
```

2. Propagation Accuracy

```
# Current (assuming):
pos = sgp4_propagate(tle, time)

# Need uncertainty:
pos, covariance = sgp4_with_uncertainty(tle, time)
# Older TLEs → higher uncertainty
# This is critical for collision probability
```

3. Risk Score Calibration

```
# Current:
risk = exp(-distance / scale)  # Arbitrary scale

# Need calibrated probability:
risk = collision_probability(
    distance,
    rel_velocity,
    covariance,
    object_sizes
)
# Should match observed collision rates
```

4. Temporal Dimension

```
# Current: Evaluates current state

# Need: Forecast risk over time
risk_timeline = []
for t in range(0, 7*24):  # 7 days hourly
    pairs_at_t = find_close_approaches(satellites, t)
    risk_timeline.append(max_risk_at_t)

# Plot risk evolution
# Alert when risk spike detected
```

Product Positioning (Based on What You Have)

Current State: "Risk Radar Dashboard"

Pitch: "Real-time proximity monitoring for satellite operators"

Target users:

- Small satellite operators (5-20 satellites)
- University CubeSat programs
- Emerging space companies

Price point: $200-500/month

Value prop:

- See close approaches instantly
- No $100k AGI contract needed
- Modern UI, not 1990s software

Next Phase: "Predictive Collision Intelligence"

Pitch: "ML-powered risk forecasting for orbital safety"

Target users:

- Mid-size constellations (20-100 satellites)
- Space insurance underwriters
- Constellation operators

Price point: $1k-5k/month + API fees

Value prop:

- 7-day risk forecasts
- GNN-based scoring (better than distance alone)
- Historical validation (proven accuracy)

Go-to-Market Strategy

Phase 1: Free Beta (Now)

1. **Post on Reddit** (r/space, r/SpaceXLounge)
    - "I built a free satellite collision risk dashboard"
    - Link to demo
    - Collect feedback
2. **Reach out to universities**
    - MIT, Stanford, Purdue (have CubeSat programs)
    - "Free tool for your constellation monitoring"
    - Build case studies
3. **X/Twitter presence**
    - Space community is very active there
    - Share cool visualizations

    - Tag relevant accounts (@planet4589, @marco_langbroek)

Phase 2: Validate with Real Users (2-3 months)

1. Get 10 beta users using it weekly
2. Collect feedback on accuracy
3. Validate against real conjunctions
4. Build credibility: "Caught 15/20 close approaches in our beta"

Phase 3: Monetize (Month 4+)

1. Launch paid tier: $200/month
    - Higher update frequency
    - More satellites tracked
    - CSV export + API access
2. Insurance pilot:
    - Approach 2-3 space insurers
    - "Risk scoring API for underwriting"
    - $5k/year pilot contracts

Competitive Differentiation (Now)

vs. AGI Comspoc:

- ✅ 500x cheaper ($0 vs $250k)
- ✅ Modern UI (web-based vs desktop)
- ✅ Instant access (no sales call)
- ❌ Less accurate (no radar data)
- ❌ Limited satellites (vs 40k objects)

vs. JSpOC (free government alerts):

- ✅ Better UX (visual vs email alerts)
- ✅ Configurable (set your own thresholds)
- ✅ Historical analysis (not just real-time)
- ❌ Same data source (NORAD TLEs)

vs. LeoLabs:

- ✅ Free tier (vs $10k+)
- ✅ Open methodology (vs black box)
- ❌ No proprietary radar data
- ❌ Limited coverage

Your wedge: Accessibility + transparency

Honest Assessment

What you've accomplished:

This is a solid **MVP**. Many founders don't get this far.

What's missing for revenue:

1. **Validation** - Prove accuracy with real conjunctions
2. **ML upgrade** - Distance alone isn't enough long-term
3. **API** - Operators want programmatic access
4. **Scale** - Need more satellites tracked

Timeline to first dollar:

- **Aggressive:** 2 months (find beta users, launch $200/month tier)
- **Realistic:** 4-6 months (validate, add ML, build credibility)
- **Conservative:** 12 months (full GNN, insurance API, enterprise features)

My Recommendation

Option A: Product Route (Recommended)

Focus: Get 5 paying customers in 3 months

Steps:

1. This week: Post beta publicly, collect users
2. Week 2-4: Validate accuracy with CDM data
3. Month 2: Add ML features (LightGBM risk model)
4. Month 3: Launch $200/month tier
5. Measure: Did anyone pay?

Option B: Research Route

Focus: Build best GNN model, publish paper, get acquired

Steps:

1. Month 1-2: Implement GNN properly
2. Month 3-4: Validate on large dataset
3. Month 5-6: Write paper, submit to conference
4. Show to AGI/LeoLabs for acquisition

Option C: Hybrid (My vote)

1. Get beta users NOW (validate market)
2. Add ML in parallel (improve product)
3. Revenue + credibility → fundraising or acquisition

Immediate Action Items (This Weekend)

1. **Public launch:**
2. `- Post on r/space: "I built a free satellite risk radar"`
3. `- Post on HackerNews: "Show HN: Real-time satellite collision monitoring"`
4. `- Tweet it with demo link`
5. **Get Space-Track CDM data:**
6. `# Download past 30 days of CDMs`
7. `# Compare your top risk pairs vs actual conjunctions`
8. `# Calculate precision/recall`
9. **Add one killer feature:**
    - **Timeline view:** Show how risk evolves over 7 days
    - OR **Animated orbits:** Show satellites moving in 3D
    - OR **Alert system:** Email when risk > threshold
10. **Create landing page:**
    - Explain what it does
    - Show demo screenshots
    - Collect emails for beta

Final Verdict

You're further along than 95% of "idea stage" founders.

You have:

- ✅ Working product
- ✅ Real data
- ✅ Clean UI
- ✅ Technical chops

You need:

- ⚠️ User validation (does anyone want this?)
- ⚠️ Accuracy validation (is it reliable?)
- ⚠️ Revenue model (will anyone pay?)

Next critical milestone:
Get 10 people using this weekly. Ask them: "Would you pay $200/month for this?"

If 3+ say yes → you have a business.
If 0 say yes → pivot to insurance API or research angle.

Want me to help with:

1. Validation script (compare your predictions vs CDM ground truth)?
2. Public launch copy (Reddit/HN posts)?
3. Feature prioritization (what to build next)?
4. Pricing strategy?

This is legit. Don't let it die in a GitHub repo. Ship it publicly this week.