SuperGrokSnipV1 - COMPLETE IMPLEMENTATION TASK LIST

Verified & Production-Ready Checklist

📋 PRE-DEVELOPMENT TASKS

Environment Setup

- [ ] Install Python 3.10 or 3.11 (verify: `python --version`)
- [ ] Install Visual Studio Build Tools (Windows only)
- [ ] Install Node.js LTS (verify: `node --version`)
- [ ] Install Rust (verify: `rustc --version`)
- [ ] *Alternative*: Setup WSL2 Ubuntu 22.04 (recommended)
- [ ] Install Git and configure SSH keys
- [ ] Clone/create project repository

API Keys & Accounts

- [ ] Create Helius account (FREE: dev.helius.xyz)
- [ ] Copy API key to `.env`
- [ ] Note: 500K credits/month, 10 req/sec
- [ ] Create BSCScan account (FREE: bscscan.com/apis)
- [ ] Copy API key to `.env`
- [ ] Create Telegram bot (@BotFather)
- [ ] Copy bot token to `.env`
- [ ] *Optional*: Twitter API v2 ($100/mo)
- [ ] Or use ntscraper (free)
- [ ] Create Solana wallet (Phantom, Solflare)
- [ ] Copy private key to `.env` (**ENCRYPT**)
- [ ] Fund with 0.1-1 SOL for testing

Repository Setup

- [ ] Create folder structure (see DELIVERABLE PACKAGE)
- [ ] Copy configuration files from artifacts
- [ ] Run `pip install -r requirements.txt`
- [ ] Run `npm install` in dashboard folder
- [ ] Copy `.env.example` to `.env` and fill values
- [ ] Test imports: `python -c "import solana, torch, prophet"`

🔨 PHASE 1: FOUNDATION (Days 1-7)

Day 1-2: Core Infrastructure

- [ ] **Test RPC Connections**

```python
# Solana
from solana.rpc.api import Client
client = Client("https://api.mainnet-beta.solana.com")
print(client.is_connected())

# BSC
from web3 import Web3
w3 = Web3(Web3.HTTPProvider("https://bsc-dataseed.binance.org"))
print(w3.is_connected())
```

- [ ] **Implement Logger** (`utils/logger.py`)
- [ ] JSON format
- [ ] Separate files: bot.log, trades.log, errors.log
- [ ] Test logging to all files
- [ ] **Implement Config Loader** (`config_loader.py`)
- [ ] Load YAML configs
- [ ] Environment variable substitution
- [ ] Validation for required fields
- [ ] **Test Helius Webhook** (optional)
- [ ] Create webhook via UI
- [ ] Setup ngrok for localhost testing
- [ ] Receive test event

Day 3-4: Monitoring Module

- [ ] **Solana Monitor** (`modules/monitor_solana.py`)
- [ ] WebSocket subscription to Raydium V4

```python
RAYDIUM_V4 = "675kPX9MHTjS2zt1qfr1NYHuzeLXfQM9H24wFSUt1Mp8"
PUMP_FUN = "6EF8rrecthR5Dkzon8Nwu78hRvfCKubJ14M5uBEwF6P"
```

- [ ] Parse "initialize2" events
- [ ] Extract token address from logs
- [ ] Fallback polling (400ms)
- [ ] Test: Detect 10 new pools on devnet
- [ ] **BSC Monitor** (`modules/monitor_bsc.py`)

- [ ] Event filter for PancakeSwap PairCreated
- [ ] Parse token0, token1, pair address
- [ ] Filter for BNB pairs only
- [ ] Test: Detect 10 new pairs on testnet
- [ ] **Pre-Filters**
- [ ] Minimum liquidity check ($5K)
- [ ] Minimum holders (50+)
- [ ] Token age (< 5 minutes)
- [ ] Top 10 ownership (<60%)
- [ ] Log all filtered tokens

Milestone 1 Test: Run for 1 hour, detect 20+ new tokens

Day 5-6: Safety Filters

- [ ] **RugCheck Integration** (`modules/safety_rugcheck.py`)

```python
async def check_rugcheck(token_address: str) -> dict:
    url = f"https://api.rugcheck.xyz/v1/tokens/{token_address}/report"
    # GET request, parse score (0-10)
    # Return: {'is_safe': bool, 'score': int}
```

- [ ] Implement caching (5 min TTL)
- [ ] Handle rate limits (429 errors)
- [ ] Test with 20 known tokens
- [ ] **Honeypot Detection** (`modules/safety_honeypot.py`)
- [ ] Install wrapper: `pip install honeypot-is`
- [ ] Test BSC honeypot detection
- [ ] Check buy_tax, sell_tax thresholds
- [ ] **On-Chain Checks**
- [ ] Mint authority check (must be None)
- [ ] Freeze authority check (must be None)
- [ ] LP burn verification (≥5%)
- [ ] LP lock check via Solscan/BSCScan
- [ ] **Threat Detection**
- [ ] Vampire attack (LP migrated < 1h)
- [ ] PvP warning (dev holds >20%)
- [ ] Bot swarm (100+ first block txs)

Milestone 2 Test: 95%+ rug filter accuracy on 100 tokens

Day 7: Integration Testing

- [ ] Combine monitor + safety filters
- [ ] Test full pipeline on testnet:
- [ ] Detect new token
- [ ] Run all safety checks
- [ ] Log decision (SNIPE/SKIP)
- [ ] Record false positives/negatives
- [ ] Adjust thresholds based on results
- [ ] Document detection latency (<500ms)

Deliverable: Bot detects launches, filters 95% of rugs

🧠 PHASE 2: INTELLIGENCE (Days 8-14)

Day 8-9: Dev Wallet Analyzer

- [ ] **Transaction Parser** (`modules/dev_analyzer.py`)
- [ ] Extract dev address from token creator
- [ ] Fetch 500 transactions via Helius/BSCScan
- [ ] Parse token creation events
- [ ] Identify rugs (<4h dumps)
- [ ] Identify pumps (24-72h holds, >50% gains)
- [ ] **Scoring Algorithm**

```python
score = 50  # Baseline
score += pumps * 50
score -= rugs * 30
score += 30 if balance > 100 SOL else -20
score += 20 if connected_to_kols else 0
# Clamp: 0-100
```

- [ ] Implement NetworkX wallet graph
- [ ] Estimate total wealth (direct + connected)
- [ ] Test on 20 devs (10 known ruggers, 10 successful)
- [ ] **ML Classifier** (Optional)
- [ ] Collect 100 labeled dev wallets
- [ ] Features: [rugs, pumps, balance, age, connections]
- [ ] Train RandomForest
- [ ] Save to `data/models/dev_classifier.pkl`
- [ ] Test accuracy (target: 75%+)

Milestone 3 Test: 80%+ accuracy classifying devs

Day 10-11: LEP (Liquidity Event Predictor)

- [ ] **Signal Detection** (`modules/lep.py`)
- [ ] Funding spike detector

```python
# Check if dev receives 10x avg transfer
recent_transfers = get_transfers(dev_address, limit=10)
avg = mean([t.amount for t in recent_transfers])
if latest_transfer > avg * 10:
    score += 40  # CEX listing fee likely
```

- [ ] Contract upgrade monitor
- [ ] Solana: `Realloc` instruction
- [ ] BSC: Proxy `implementation()` changes
- [ ] Coordinated accumulation
- [ ] Find 5+ wallets buying within 2h
- [ ] Social-chain fusion
- [ ] Scrape dev tweets (hype keywords)
- [ ] Correlate with wallet velocity
- [ ] **Prophet Training**

```python
from prophet import Prophet

# Historical data: 50+ past CEX listings
df = pd.DataFrame({
    'ds': timestamps,  # Signal detection time
    'y': hours_until_pump,
    'funding_spike': [0 or 1],
    'contract_upgrade': [0 or 1],
    'accumulation': [0-100],
    'social_hype': [0-100]
})

model = Prophet()
model.add_regressor('funding_spike')
# ... add others
model.fit(df)
model.save('data/models/lep_prophet.pkl')
```

- [ ] Collect 50-100 historical tokens
- [ ] Label: hours from signal to pump
- [ ] Train model (10 epochs, ~5 min)
- [ ] Backtest on 10 tokens

Milestone 4 Test: 60%+ detection of pumps 6-48h early

Day 12-13: Cascade Sentinel (Viral Predictor)

- [ ] **Twitter Scraping**
- [ ] Choose: ntscraper (free) or official API ($100/mo)
- [ ] Scrape 100 recent tweets mentioning token
- [ ] Extract: user, text, retweets, likes, followers
- [ ] **Graph Building** (`utils/graph_builder.py`)

```python
import networkx as nx

G = nx.DiGraph()
for tweet in tweets:
    G.add_node(tweet.user, followers=tweet.user.followers)
    for mention in tweet.mentions:
        G.add_edge(tweet.user, mention, weight=1)

# Convert to PyTorch Geometric
from torch_geometric.data import Data
data = Data(x=node_features, edge_index=edges)
```

- [ ] **GNN Training** (`ai/cascade_gnn.py`)
- [ ] Collect 50+ historical tokens
- [ ] Build graph for each
- [ ] Label: 1 if pumped 10x+, 0 if not
- [ ] Train GCN (3 layers, 50 epochs)

```python
class CascadeGNN(torch.nn.Module):
    def __init__(self):
        self.conv1 = GCNConv(3, 16)
        self.conv2 = GCNConv(16, 32)
        self.conv3 = GCNConv(32, 1)
```

- [ ] Save to `data/models/cascade_gnn.pt`
- [ ] Test accuracy (target: 70%+)

Milestone 5 Test: 70%+ virality prediction accuracy

Day 14: Integration Testing

- [ ] Test full AI pipeline:
- [ ] New token detected
- [ ] Dev analyzer: Score 82/100 ✅
- [ ] LEP: Score 165/200 ✅
- [ ] Cascade: Score 78/100 ✅
- [ ] Decision: SNIPE NOW
- [ ] Test on 10 historical tokens
- [ ] Validate against known outcomes
- [ ] Document win rate, false positives

Deliverable: AI predicting pumps 6-48h early

🚀 PHASE 3: PRODUCTION (Days 15-21)

Day 15-16: Execution Engine

- [ ] **Jito Bundle Creation** (`modules/execution_jito.py`)

```python
from jito_searcher_client import get_searcher_client

async def send_jito_bundle(swap_tx, tip=0.002):
    client = get_searcher_client("mainnet")
    tip_tx = create_tip_to_jito(wallet, tip)
    bundle = [swap_tx, tip_tx]
    return await client.send_bundle(bundle)
```

- [ ] Get Jito tip account (rotates per epoch)
- [ ] Build swap transaction
- [ ] Build tip transaction
- [ ] Bundle and submit
- [ ] Test on devnet (simulate)
- [ ] **BSC High-Gas Strategy** (`modules/execution_bsc.py`)
- [ ] Calculate optimal gas (1.5x base)
- [ ] Use Eden RPC if available
- [ ] Implement retry with exponential backoff
- [ ] Test on testnet
- [ ] **Dynamic Slippage**

```python
def calculate_slippage(token_volatility):
    if volatility < 0.10: return 0.01
    elif volatility < 0.30: return 0.03
    elif volatility < 0.50: return 0.05
    else: return None  # Too risky
```

- [ ] **Auto-Sell** (`modules/execution.py`)
- [ ] Monitor position PnL
- [ ] Take profit at 3x (configurable)
- [ ] Stop loss at -20%
- [ ] Trailing stop (20% from peak)
- [ ] Dev dump detection (sell if dev dumps >30%)
- [ ] Force sell after 24h

Milestone 6 Test: 90%+ transaction land rate with Jito

Day 17-18: React Dashboard

- [ ] **Setup Next.js + Tailwind**

```bash
cd src/ui/dashboard
npx create-next-app@latest .
npm install tailwindcss recharts lucide-react zustand
```

- [ ] **Main Components**
- [ ] `Dashboard.tsx`: Balance, profit, active snipes
- [ ] `Settings.tsx`: Investment config, safety toggles
- [ ] `Monitor.tsx`: Real-time token feed
- [ ] `History.tsx`: Trade log with PnL
- [ ] `AutoTrainCard.tsx`: Retrain button + alerts
- [ ] `Charts.tsx`: Performance over time (Recharts)
- [ ] **Flask Backend** (`ui/server.py`)

```python
```

```
from flask import Flask, jsonify
from flask_socketio import SocketIO

app = Flask(__name__)
socketio = SocketIO(app, cors_allowed_origins="*")

@app.route('/api/status')
def status():
    return jsonify({'active': True, 'balance': 250})

@socketio.on('connect')
def handle_connect():
    # Push real-time updates
    pass
```

- [ ] **Websocket Integration**
- [ ] Push new token detections
- [ ] Push trade executions
- [ ] Push AI scores in real-time
- [ ] **Styling** (Black/White theme)
- [ ] Tailwind config from artifacts
- [ ] Dark mode with neon accents
- [ ] Responsive (desktop + mobile)

Milestone 7 Test: Dashboard shows live data

Day 19-20: Auto-Training System

- [ ] **MLflow Setup**

```
bash

pip install mlflow
mlflow server --host 127.0.0.1 --port 5000
```

- [ ] Create experiment: `SuperGrokSnipV1_Training`
- [ ] Test logging: model, metrics, artifacts
- [ ] **Data Collector** (`ai/data_collector.py`)
- [ ] DexScreener API for historical tokens
- [ ] Helius webhook for real-time
- [ ] Twitter scraper for social data
- [ ] Auto-label: pump vs rug (48h outcome)
- [ ] Save to `data/historical_pumps/`
- [ ] **Smart Retraining Engine** (`ai/smart_retrain.py`)

```python
def should_retrain():
    score = 0
    if lep_accuracy < 0.65: score += 30
    if data_age > 7: score += 25
    if drift > 0.4: score += 25
    if volatility > 1.5: score += 20
    return score >= 50  # Recommend retrain
```

- [ ] Implement drift detection (KS test)
- [ ] Market volatility monitor
- [ ] Generate alert with reasons
- [ ] **Training Orchestrator** (`ai/auto_train.py`)

```python
def run_pipeline():
    # 1. Collect 50-100 fresh samples
    df = auto_collect_data()
    # 2. Validate & split
    X_train, X_val = split_data(df)
    # 3. Train all models
    models = train_models(X_train)
    # 4. Evaluate
    accuracies = evaluate(models, X_val)
    # 5. Log to MLflow
    with mlflow.start_run():
        for name, model in models.items():
            mlflow.log_model(model, name)
    # 6. Deploy (update config paths)
    deploy_models()
```

- [ ] Test manual trigger
- [ ] Test smart alert trigger
- [ ] Verify model updates

Milestone 8 Test: Retrain all models in <30 min

Day 21: Final Integration

- [ ] Run full system end-to-end:
- [ ] Monitor detects new token
- [ ] Safety filters applied
- [ ] Dev analyzer scores

- [ ] LEP predicts timing
- [ ] Cascade predicts virality
- [ ] Execution via Jito
- [ ] Auto-sell at 3x
- [ ] UK CGT calculated
- [ ] Dashboard updates
- [ ] Telegram alert sent
- [ ] **Paper Trading Test** (testnet)
- [ ] Simulate 100 trades
- [ ] Record: wins, losses, latency
- [ ] Target: 60%+ win rate
- [ ] **Performance Benchmarks**
- [ ] Detection latency: <500ms
- [ ] Full analysis: <2s
- [ ] Execution: <200ms (Jito)
- [ ] Total: Token detected to trade executed <3s

Deliverable: Production-ready bot with UI

🎯 PHASE 4: REFINEMENT (Days 22-28)

Day 22-23: Optimization

- [ ] **Caching Layer**
- [ ] Redis for API responses (5 min TTL)
- [ ] LRU cache for token checks
- [ ] Reduce API calls by 70%
- [ ] **Multi-Wallet Rotation** (`modules/wallet_manager.py`)
- [ ] Generate 10 burner wallets
- [ ] Fund each with 0.1 SOL/BNB
- [ ] Rotate after each trade
- [ ] Track balances across wallets
- [ ] **Rate Limit Handling**

```python
```

```python
@retry(wait=exponential(min=1, max=60), stop=stop_after_attempt(3))
async def api_call_with_retry(url):
    response = await aiohttp.get(url)
    if response.status == 429:
        await asyncio.sleep(int(response.headers['Retry-After']))
        raise Exception("Rate limited")
    return response
```

Day 24-25: Security

- [ ] **Private Key Encryption**

```python
from cryptography.fernet import Fernet

key = Fernet.generate_key()  # Save to secure location
cipher = Fernet(key)
encrypted_pk = cipher.encrypt(private_key.encode())
# Store encrypted_pk in .env
```

- [ ] **Drain Detection**
- [ ] Monitor wallet balance every 30s
- [ ] Alert if >50% drained unexpectedly
- [ ] Auto-pause bot on suspicious activity
- [ ] **Audit Log**
- [ ] Log ALL trades with signatures
- [ ] Immutable append-only log
- [ ] Periodic backup to S3/cloud
- [ ] **Security Checklist**
- [ ] `.env` in `.gitignore`
- [ ] Private keys encrypted at rest
- [ ] API keys rotated every 90 days
- [ ] 2FA on all accounts
- [ ] Regular security audits

Day 26-27: Documentation

- [ ] **User Manual** (`docs/USER_MANUAL.md`)
- [ ] Getting started guide
- [ ] Configuration walkthrough
- [ ] Dashboard tutorial
- [ ] FAQ section

- [ ] **API Reference** (`docs/API_REFERENCE.md`)
- [ ] Module documentation
- [ ] Function signatures
- [ ] Example usage
- [ ] Error codes
- [ ] **Troubleshooting** (`docs/TROUBLESHOOTING.md`)
- [ ] Common issues
- [ ] RPC failures
- [ ] Transaction errors
- [ ] ML model issues
- [ ] **Video Tutorials**
- [ ] Setup walkthrough (15 min)
- [ ] Configuration guide (10 min)
- [ ] First trade demo (5 min)

Day 28: Mainnet Deployment

- [ ] **Pre-Launch Checklist**
- [ ] All tests passing
- [ ] 100+ paper trades successful
- [ ] Dashboard functional
- [ ] Alerts working
- [ ] Backups configured
- [ ] Kill switch tested
- [ ] **Gradual Rollout**
- [ ] Start: $10-$20 per trade
- [ ] Monitor for 24h
- [ ] Increase: $50 per trade
- [ ] Monitor for 48h
- [ ] Increase: $100+ per trade
- [ ] **Monitoring**
- [ ] Real-time dashboard
- [ ] Telegram alerts
- [ ] Error logging
- [ ] Performance metrics
- [ ] **Emergency Procedures**
- [ ] Kill switch: Stop all trading
- [ ] Withdraw all funds
- [ ] Investigate issues
- [ ] Resume when resolved

Final Deliverable: Live bot on mainnet

✅ VERIFICATION CHECKLIST

Code Quality

- [ ] All modules have docstrings
- [ ] Type hints for all functions
- [ ] PEP 8 compliant (black + flake8)
- [ ] No hardcoded credentials
- [ ] Error handling for all API calls
- [ ] Logging for all critical operations

Testing

- [ ] Unit tests: 80%+ coverage
- [ ] Integration tests passing
- [ ] Backtests: 60%+ win rate
- [ ] Paper trading: 100 successful trades
- [ ] Load testing: Handle 100 tokens/hour

Security

- [ ] Private keys encrypted
- [ ] `.env` not committed
- [ ] API keys in environment variables
- [ ] No sensitive data in logs
- [ ] Regular security audits scheduled

Documentation

- [ ] README complete
- [ ] All configs documented
- [ ] User manual written
- [ ] API reference generated
- [ ] Video tutorials recorded

Performance

- [ ] Detection latency <500ms
- [ ] Full analysis <2s
- [ ] Transaction execution <200ms
- [ ] Dashboard updates <1s
- [ ] API rate limits handled

Legal & Compliance

- [ ] UK CGT auto-calculated
- [ ] Trade logs exportable
- [ ] GDPR compliance (if applicable)
- [ ] Terms of service drafted
- [ ] Risk disclaimers displayed

📊 SUCCESS METRICS

Technical

- **Detection Rate**: 95%+ of new tokens caught
- **Filter Accuracy**: 95%+ rugs blocked
- **Transaction Success**: 90%+ land rate (Jito)
- **Latency**: <3s from detection to execution
- **Uptime**: 99%+ (excluding maintenance)

Financial

- **Win Rate**: 60%+ of trades profitable
- **ROI**: 2-5x vs standard sniping
- **Max Drawdown**: <10% of capital
- **Daily Loss Limit**: <5% triggered
- **Tax Reporting**: 100% accurate for HMRC

Operational

- **Auto-Training**: Models retrain when alerted
- **Alerts**: 100% delivered (Telegram)
- **Dashboard**: Real-time updates (<1s lag)
- **Logs**: Complete audit trail
- **Backups**: Daily automated

🎓 LESSONS LEARNED

From Research

1. **Jito mempool shut down** - Focus on atomic execution, not frontrunning
2. **Prophet easier than LSTM** - Faster training, good for 48h forecasts
3. **Free tier APIs sufficient** - For testing; upgrade for production
4. **LEP is the edge** - 6-48h early entry = 40x vs 5x

Best Practices

1. **Start simple** - Monitor → Filters → Execution (skip AI initially)
2. **Test extensively** - 100+ paper trades before mainnet
3. **Log everything** - Essential for debugging and training data
4. **Incremental scaling** - $10 → $50 → $100 → $500

Common Pitfalls

1. **Over-optimization** - 70-80% accuracy is production-ready
2. **Feature creep** - Ship MVP first, add LEP/Cascade later
3. **Ignoring false positives** - Test on 100+ tokens to find edge cases
4. **Underestimating time** - Allow 6 weeks, not 4 weeks

🏁 FINAL CHECKLIST

- [ ] All Phase 1-4 tasks completed
- [ ] 100+ paper trades successful
- [ ] Dashboard operational
- [ ] Auto-training working
- [ ] Documentation complete
- [ ] Security audited
- [ ] Mainnet deployed with $10-50
- [ ] 24h monitoring successful
- [ ] Ready to scale

STATUS: Ready to build! 🚀

Estimated Total Time: 4-6 weeks (full-time)
Investment: $0 (free tier) to $200/mo (paid APIs)

Expected ROI: 2-5x vs standard sniping
Risk Level: Medium-High (crypto volatility)

Good luck! Follow this checklist methodically and you'll have a production-ready sniper bot.