Detailed Implementation Guide: VLMS Companion Formation Pathways Enhancement

Implementation Overview

This guide provides step-by-step instructions for integrating the new analysis features into the existing VLMS Companion Analysis System codebase while maintaining backward compatibility.

Phase 1: Infrastructure Updates

1.1 Dependencies Update

File to modify: `requirements.txt`

Add the following new dependencies:

```
# Existing dependencies remain unchanged
# Add these new dependencies:
hdbscan>=0.8.27
ruptures>=1.1.5
```

1.2 Command-Line Interface Extension

File to modify: `source/panoptic_vlms_project.py`

Add new command-line arguments to the existing `parse_args()` function:

```
# After existing argument definitions (around line 50-100), add:

# Disk migration parameters
parser.add_argument("--disk-panel", action="store_true",
                    help="Render disk-migration timescale panel alongside KL")
parser.add_argument("--disk-lifetime-myr", type=float, default=3.0,
                    help="Target disk lifetime for feasible migration (Myr)")
parser.add_argument("--a0-min", type=float, default=0.3,
                    help="Minimum birth a0 (AU) to sweep in disk panel")
parser.add_argument("--a0-max", type=float, default=1.0,
                    help="Maximum birth a0 (AU) to sweep in disk panel")
```

```
parser.add_argument("--Sigma1AU", type=float, default=300.0,
                    help="Gas surface density at 1 AU (g/cm^2) for VLMS disk")
parser.add_argument("--p-sigma", type=float, default=1.0,
                    help="Surface-density power-law: Sigma ~ a^{-p}")
parser.add_argument("--H-over-a", type=float, default=0.04,
                    help="Disk aspect ratio H/a")
parser.add_argument("--alpha", type=float, default=3e-3,
                    help="Viscosity parameter for gap-opening check")

# Enhanced KL parameters
parser.add_argument("--kl-a0", type=float, default=0.5,
                    help="Birth inner a0 (AU) used by KL feasibility map")
parser.add_argument("--kl-horizon-gyr", type=float, default=3.0,
                    help="Time horizon (Gyr) for KL+tides feasibility")
parser.add_argument("--rpcrit-Rs", type=float, default=3.0,
                    help="Critical periastron in stellar radii")

# System-level analysis flags
parser.add_argument("--build-systems", action="store_true",
                    help="Aggregate per-companion tables into system-level rows")
parser.add_argument("--sb-csv", type=str, default=None,
                    help="Path to CSV of close VLMS stellar binaries")
parser.add_argument("--regimes", action="store_true",
                    help="Run HDBSCAN+GMM clustering and segmented regression")
parser.add_argument("--msr", action="store_true",
                    help="Run 2-regime mixture of linear regressions")
```

Phase 2: Core Module Implementations

2.1 Disk Migration Module

New file to create: `source/disk_migration.py`

```
"""
Disk migration timescale calculations for Type-I migration
"""
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from pathlib import Path

# Physical constants
AU = 1.495978707e13  # cm
M_sun = 1.98847e33   # g
M_jup = 1.89813e30   # g
G = 6.67430e-8       # cgs
YEAR = 3.15576e7     # s

def _Omega(Mstar_Msun, a_AU):
    """Calculate orbital frequency"""
    M = Mstar_Msun * M_sun
    a = a_AU * AU
```

```python
    return np.sqrt(G * M / a**3)

def typeI_timescale_sec(Mstar_Msun, Mp_Mj, a_AU, Sigma1_gcm2, p_sigma, H_over_a, C=3.0):
    """
    Calculate Type-I migration timescale using Tanaka et al. formalism

    Parameters:
    -----------
    Mstar_Msun : float
        Stellar mass in solar masses
    Mp_Mj : float
        Planet mass in Jupiter masses
    a_AU : float
        Semimajor axis in AU
    Sigma1_gcm2 : float
        Surface density at 1 AU in g/cm^2
    p_sigma : float
        Surface density power law index
    H_over_a : float
        Disk aspect ratio
    C : float
        Calibration constant (default 3.0)

    Returns:
    --------
    float : Migration timescale in seconds
    """
    Mstar = Mstar_Msun * M_sun
    Mp = Mp_Mj * M_jup
    a = a_AU * AU
    Sigma = Sigma1_gcm2 * (a_AU ** (-p_sigma))
    Omega = _Omega(Mstar_Msun, a_AU)

    return C * (Mstar/Mp) * (Mstar/(Sigma * a*a)) * (H_over_a**2) / Omega

def migrate_time_numeric(Mstar_Msun, Mp_Mj, a0_AU, af_AU,
                         Sigma1_gcm2, p_sigma, H_over_a, C=3.0, nstep=2000):
    """
    Numerically integrate migration time from a0 to af

    Returns:
    --------
    float : Total migration time in seconds
    """
    a_hi, a_lo = max(a0_AU, af_AU), min(a0_AU, af_AU)
    a_grid = np.geomspace(a_hi, a_lo, nstep)

    vals = []
    for a in a_grid:
        t_local = typeI_timescale_sec(Mstar_Msun, Mp_Mj, a,
                                      Sigma1_gcm2, p_sigma, H_over_a, C)
        vals.append(t_local / (a * AU))

    dt_sec = np.trapz(vals, a_grid)
    return dt_sec
```

```python
def render_disk_panel(outpath_png, Mstar_Msun, Mp_Mj, args):
    """
    Create heatmap of migration timescales

    Parameters:
    -----------
    outpath_png : str
        Output path for figure
    Mstar_Msun : float
        Host star mass
    Mp_Mj : float
        Companion mass in Jupiter masses
    args : namespace
        Command-line arguments containing disk parameters
    """
    a0s = np.geomspace(args.a0_min, args.a0_max, 60)
    Sigma1_list = np.geomspace(args.Sigma1AU/5.0, args.Sigma1AU*5.0, 60)
    Z = np.zeros((len(Sigma1_list), len(a0s)))

    af = 0.05  # Final position in AU

    for i, S1 in enumerate(Sigma1_list):
        for j, a0 in enumerate(a0s):
            t_sec = migrate_time_numeric(Mstar_Msun, Mp_Mj, a0, af,
                                         S1, args.p_sigma, args.H_over_a, C=3.0)
            Z[i, j] = t_sec / (1e6 * YEAR)  # Convert to Myr

    fig, ax = plt.subplots(1, 1, figsize=(7.2, 5.4), constrained_layout=True)

    im = ax.pcolormesh(a0s, Sigma1_list, Z, shading='auto',
                       norm=mpl.colors.LogNorm(vmin=0.1, vmax=100))
    cb = fig.colorbar(im, ax=ax, label="Migration time (Myr) to 0.05 AU")

    # Add disk lifetime contour
    cs = ax.contour(a0s, Sigma1_list, Z, levels=[args.disk_lifetime_myr],
                    colors='white', linewidths=1.5)
    ax.clabel(cs, fmt={args.disk_lifetime_myr: f"{args.disk_lifetime_myr:.0f} Myr"},
              inline=True)

    ax.axhline(args.Sigma1AU, ls='--', lw=1, color='k', alpha=0.5)
    ax.set_xscale('log')
    ax.set_yscale('log')
    ax.set_xlabel(r"Birth $a_0$ (AU)")
    ax.set_ylabel(r"$\Sigma_{1\,\mathrm{AU}}$ (g cm$^{-2}$)")
    ax.set_title("Disk Migration: Is $a_0\\rightarrow0.05$ AU feasible?")

    # Add parameter box
    ax.text(0.03, 0.02,
            f"H/a={args.H_over_a:.2f}, p={args.p_sigma:.1f}, α={args.alpha:.0e}\n"
            f"M*={Mstar_Msun:.2f} M$_\\odot$, M$_c$={Mp_Mj:.2f} M$_J$",
            transform=ax.transAxes, fontsize=9, va='bottom')

    fig.savefig(outpath_png, dpi=200)
    plt.close(fig)
    return Z
```

2.2 System-Level Data Schema

New file to create: `source/system_schema.py`

```python
"""
System-level data aggregation for multi-companion analysis
"""
import numpy as np
import pandas as pd
from pathlib import Path

MJUP_PER_MSUN = 1047.56

def _log10_safe(x):
    """Safe log10 that handles zeros and negatives"""
    x = np.asarray(x, dtype=float)
    with np.errstate(divide="ignore", invalid="ignore"):
        y = np.log10(x)
    return y

def build_system_table(nasa_csv="results/pscomppars_lowM.csv",
                       bd_csv="results/BD_catalogue.csv",
                       sb_csv=None,
                       out_csv="results/combined_systems.csv"):
    """
    Aggregate companion data into system-level rows

    Parameters:
    -----------
    nasa_csv : str
        Path to NASA Exoplanet Archive data
    bd_csv : str
        Path to Brown Dwarf catalog
    sb_csv : str, optional
        Path to stellar binary catalog
    out_csv : str
        Output path for combined system table

    Returns:
    --------
    pd.DataFrame : System-level aggregated data
    """
    # Load NASA data
    nasa = pd.read_csv(nasa_csv)

    # Normalize column names (handle variations)
    host_col = next((c for c in nasa.columns
                    if c.lower() in {"hostname", "pl_hostname", "sy_name"}), None)
    if host_col is None:
        raise ValueError("Could not find host name column in NASA CSV")

    # Map columns to standard names
    nasa = nasa.rename(columns={host_col: "host"})

    # Process companion data
```

```python
all_comp = [nasa]

# Add brown dwarf data if available
try:
    bd = pd.read_csv(bd_csv)
    all_comp.append(bd)
except FileNotFoundError:
    pass

# Add stellar binaries if provided
if sb_csv:
    from source.ingest_vlms_binaries import load_vlms_binaries
    sb = load_vlms_binaries(sb_csv)
    all_comp.append(sb)

# Combine all sources
comp = pd.concat(all_comp, ignore_index=True)

# System-level aggregation function
def summarize(group):
    """Aggregate companion properties per system"""
    if group.empty:
        return pd.Series(dtype=float)

    # Calculate mass ratios
    if 'Mstar' in group.columns and 'Mj' in group.columns:
        q = group['Mj'] / (group['Mstar'] * MJUP_PER_MSUN)
    else:
        q = pd.Series([np.nan])

    out = {
        'host': group['host'].iloc[0],
        'Mstar': np.nanmedian(group.get('Mstar', np.nan)),
        'n_comp': len(group),
        'q_max': np.nanmax(q),
        'q_med': np.nanmedian(q),
        'a_min': np.nanmin(group.get('a_AU', np.nan)),
        'a_med': np.nanmedian(group.get('a_AU', np.nan)),
        'e_max': np.nanmax(group.get('e', np.nan)),
        'e_med': np.nanmedian(group.get('e', np.nan)),
        'has_bd': bool(np.any(group.get('Mj', 0) >= 13)),
        'has_giant': bool(np.any(group.get('Mj', 0) >= 0.3)),
        'has_sb': bool('SB' in group.get('source', []))
    }

    return pd.Series(out)

# Group by host and aggregate
systems = comp.groupby('host', dropna=True).apply(summarize).reset_index(drop=True)

# Add logarithmic features
systems['logq_max'] = _log10_safe(systems['q_max'])
systems['loga_min'] = _log10_safe(systems['a_min'])

# Save results
Path(out_csv).parent.mkdir(parents=True, exist_ok=True)
```

```
    systems.to_csv(out_csv, index=False)

    return systems
```

2.3 Regime Discovery Module

New directory to create: `source/analysis/`

New file to create: `source/analysis/regime_clustering.py`

```
"""
Statistical regime discovery using clustering and mixture models
"""
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.mixture import GaussianMixture
from pathlib import Path

def run_hdbscan_and_gmm(systems_csv="results/combined_systems.csv",
                        out_prefix="results/regimes",
                        min_cluster_size=5,
                        random_state=42):
    """
    Perform HDBSCAN clustering and GMM validation

    Returns:
    --------
    tuple : (labeled_dataframe, bic_info_dict)
    """
    try:
        import hdbscan
    except ImportError:
        raise ImportError("Please install hdbscan: pip install hdbscan")

    # Load system data
    df = pd.read_csv(systems_csv).dropna(subset=["logq_max", "loga_min"])

    # Feature matrix
    feature_cols = ["logq_max", "loga_min", "e_max"]
    X = df[feature_cols].copy()

    # Handle missing values
    for col in X.columns:
        X[col] = pd.to_numeric(X[col], errors="coerce")
    X = X.fillna(X.median(numeric_only=True))

    # Standardize features
    scaler = StandardScaler()
    Xz = scaler.fit_transform(X.values)

    # HDBSCAN clustering
```

```python
clusterer = hdbscan.HDBSCAN(min_cluster_size=min_cluster_size)
labels = clusterer.fit_predict(Xz)
df["hdbscan_label"] = labels

# GMM with BIC selection
Ks = list(range(1, 7))
bics = []
gmms = []

for k in Ks:
    gmm = GaussianMixture(n_components=k, covariance_type="full",
                          random_state=random_state)
    gmm.fit(Xz)
    bics.append(gmm.bic(Xz))
    gmms.append(gmm)

# Select best model
k_best = Ks[int(np.argmin(bics))]
gmm_best = gmms[int(np.argmin(bics))]
df["gmm_label"] = gmm_best.predict(Xz)

# Save results
Path(out_prefix).parent.mkdir(parents=True, exist_ok=True)
df.to_csv(f"{out_prefix}_labels.csv", index=False)

# Create visualization
fig, ax = plt.subplots(figsize=(7, 6))

for lab in sorted(np.unique(labels)):
    if lab == -1:  # Noise points
        continue
    sel = df["hdbscan_label"] == lab
    ax.scatter(df.loc[sel, "logq_max"], df.loc[sel, "loga_min"],
              s=50, label=f"Cluster {lab}", alpha=0.7)

# Plot noise points
noise = df["hdbscan_label"] == -1
if noise.any():
    ax.scatter(df.loc[noise, "logq_max"], df.loc[noise, "loga_min"],
              s=20, c='gray', alpha=0.3, label="Unclustered")

ax.set_xlabel(r"$\log_{10}\,q_{\rm max}$")
ax.set_ylabel(r"$\log_{10}\,a_{\rm min}\,{\rm [AU]}$")
ax.legend(frameon=False, ncol=2, fontsize=9)
ax.set_title("HDBSCAN Regime Discovery")

fig.tight_layout()
fig.savefig(f"{out_prefix}_hdbscan_logq_loga.png", dpi=180)
plt.close(fig)

# BIC plot
fig2, ax2 = plt.subplots(figsize=(6, 4))
ax2.plot(Ks, bics, marker='o')
ax2.set_xlabel("GMM Components K")
ax2.set_ylabel("BIC (lower is better)")
ax2.axvline(k_best, ls='--', color='k', alpha=0.5,
```

```python
                label=f"Best K={k_best}")
    ax2.legend()
    fig2.tight_layout()
    fig2.savefig(f"{out_prefix}_gmm_bic.png", dpi=180)
    plt.close(fig2)

    return df, {"K_best": k_best, "BICs": bics}
```

New file to create: `source/analysis/segmented_trend.py`

```python
"""
Segmented regression analysis for phase-shift detection
"""
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from pathlib import Path
import json

def fit_one_break(logq, loga):
    """
    Fit piecewise linear regression with one break point

    Returns:
    --------
    tuple : (break_index, order, model_left, model_right, bic0, bic1, is_better)
    """
    try:
        import ruptures as rpt
    except ImportError:
        raise ImportError("Please install ruptures: pip install ruptures")

    x = np.asarray(logq).reshape(-1, 1)
    y = np.asarray(loga)

    # Sort by x for meaningful change-point detection
    order = np.argsort(x[:, 0])
    x, y = x[order], y[order]

    # Fit single line (0 breaks)
    ols0 = LinearRegression().fit(x, y)
    rss0 = np.sum((y - ols0.predict(x))**2)
    n, p0 = len(y), 2  # slope + intercept
    bic0 = n * np.log(rss0/n) + p0 * np.log(n)

    # Detect 1 break using PELT
    algo = rpt.Pelt(model="rbf").fit(y)
    cp = algo.predict(pen=np.log(n) * 5)  # Mild penalty

    # Check if valid break found
    k = None
    if len(cp) >= 2:
        k = cp[0]
```

```python
        if 2 <= k <= n-2:  # Keep interior breaks only
            xl, yl = x[:k], y[:k]
            xr, yr = x[k:], y[k:]

            olsL = LinearRegression().fit(xl, yl)
            olsR = LinearRegression().fit(xr, yr)

            rss1 = np.sum((yl - olsL.predict(xl))**2) + \
                   np.sum((yr - olsR.predict(xr))**2)
            p1 = 4  # 2 params per segment
            bic1 = n * np.log(rss1/n) + p1 * np.log(n)

            better = (bic1 + 2.0) < bic0  # Small safety margin

            return (k, order, olsL, olsR, bic0, bic1, better)

    return (None, order, ols0, None, bic0, None, False)

def run_segmented_plot(systems_csv="results/combined_systems.csv",
                       out_png="results/segmented_logq_loga.png"):
    """
    Create segmented regression visualization

    Returns:
    --------
    dict : Summary statistics
    """
    df = pd.read_csv(systems_csv).dropna(subset=["logq_max", "loga_min"])

    res = fit_one_break(df["logq_max"].values, df["loga_min"].values)
    k, order, mL, mR, bic0, bic1, better = res

    x = df["logq_max"].values[order]
    y = df["loga_min"].values[order]

    # Create visualization
    fig, ax = plt.subplots(figsize=(7, 6))
    ax.scatter(x, y, s=40, color="tab:blue", alpha=0.8, label="Systems")

    if k is not None and better:
        # Plot two segments
        xx = np.linspace(x.min(), x.max(), 200).reshape(-1, 1)

        # Left segment
        mask_left = xx[:, 0] <= x[k]
        if mask_left.any():
            ax.plot(xx[mask_left], mL.predict(xx[mask_left]),
                   color="tab:red", lw=2, label="Regime 1")

        # Right segment
        mask_right = xx[:, 0] >= x[k]
        if mask_right.any():
            ax.plot(xx[mask_right], mR.predict(xx[mask_right]),
                   color="tab:green", lw=2, label="Regime 2")

        # Mark break point
```

```python
        ax.axvline(x[k], ls="--", color="k", alpha=0.5,
                   label=f"Break at log q={x[k]:.2f}")
        ax.set_title(f"Segmented fit preferred (ΔBIC={bic0-bic1:.1f})")
    else:
        # Single line
        xx = np.linspace(x.min(), x.max(), 200).reshape(-1, 1)
        ax.plot(xx, mL.predict(xx), color="tab:red", lw=2, label="Single trend")
        ax.set_title("No significant break detected")

    ax.set_xlabel(r"$\log_{10}\,q_{\rm max}$")
    ax.set_ylabel(r"$\log_{10}\,a_{\rm min}\,{\rm [AU]}$")
    ax.legend(frameon=False)

    fig.tight_layout()
    fig.savefig(out_png, dpi=180)
    plt.close(fig)

    # Save summary
    summary = {
        "break_supported": bool(better),
        "break_logq": float(x[k]) if (k is not None and better) else None,
        "bic_single": float(bic0),
        "bic_two": float(bic1) if bic1 is not None else None,
    }

    json_path = out_png.replace(".png", ".json")
    with open(json_path, "w") as f:
        json.dump(summary, f, indent=2)

    return summary
```

Phase 3: Integration with Existing Pipeline

3.1 Modify KozaiLidovAnalyzer

File to modify: `source/statistical_analysis.py`

In the `KozaiLidovAnalyzer` class, update the `__init__` method to accept new parameters:

```python
class KozaiLidovAnalyzer:
    def __init__(self, data, toi_data=None,
                 inner_a0_AU=None,  # NEW: birth radius parameter
                 horizon_Gyr=None,   # NEW: time horizon
                 rpcrit_Rs=None):    # NEW: tidal radius threshold
        self.data = data
        self.toi_data = toi_data

        # Use new parameters if provided, else defaults
        self.inner_a0_AU = inner_a0_AU if inner_a0_AU is not None else 1.0
        self.horizon_Gyr = horizon_Gyr if horizon_Gyr is not None else 1.0
        self.rpcrit_Rs = rpcrit_Rs if rpcrit_Rs is not None else 3.0
```

```
        # Rest of initialization...
```

Update the `_simulate_kl_evolution` method to use the new parameters:

```
def _simulate_kl_evolution(self, inner_a_AU, outer_a_AU, outer_m_msun,
                          outer_e, inner_m_mj, host_m_msun):
    # Use self.inner_a0_AU instead of hardcoded value
    initial_a = self.inner_a0_AU  # Birth radius

    # Update time horizon
    max_time_yr = self.horizon_Gyr * 1e9

    # Update tidal threshold
    rp_threshold = self.rpcrit_Rs * host_radius_rsun

    # Rest of simulation...
```

3.2 Modify Main Pipeline

File to modify: `source/panoptic_vlms_project.py`

Add the following integration code after the existing analysis blocks (around line 300-400):

```
def main():
    # ... existing code ...

    # After existing data processing (around line 350)

    # Build system-level table if requested
    if args.build_systems:
        from source.system_schema import build_system_table

        systems = build_system_table(
            nasa_csv="results/pscomppars_lowM.csv",
            bd_csv="results/BD_catalogue.csv",
            sb_csv=args.sb_csv,  # Optional stellar binary file
            out_csv="results/combined_systems.csv"
        )
        logger.info(f"Built system table with {len(systems)} systems")

    # Run regime discovery if requested
    if args.regimes:
        from source.analysis.regime_clustering import run_hdbscan_and_gmm
        from source.analysis.segmented_trend import run_segmented_plot

        logger.info("Running regime discovery analysis...")

        labels_df, bic_info = run_hdbscan_and_gmm(
            systems_csv="results/combined_systems.csv",
            out_prefix="results/regimes"
```

```python
    )

    seg_summary = run_segmented_plot(
        systems_csv="results/combined_systems.csv",
        out_png="results/segmented_logq_loga.png"
    )

    logger.info(f"Regime discovery complete. Best GMM K={bic_info['K_best']}")
    if seg_summary['break_supported']:
        logger.info(f"Segmented regression found break at log q={seg_summary['break_logq']:.2f}")

# Run disk migration analysis if requested
if args.disk_panel:
    from source.disk_migration import render_disk_panel

    logger.info("Computing disk migration timescales...")

    # Use TOI-6894b or median system parameters
    if toi_data is not None:
        host_mass = toi_data.get('host_mass', 0.08)
        comp_mass = toi_data.get('comp_mass', 0.30)
    else:
        host_mass = data['host_mass'].median()
        comp_mass = data['comp_mass'].median()

    disk_png = os.path.join(args.outdir, "fig3_disk.png")
    Z = render_disk_panel(disk_png, host_mass, comp_mass, args)

    logger.info(f"Disk migration panel saved to {disk_png}")

    # Optionally combine with KL panel if it exists
    kl_png = os.path.join(args.outdir, "fig3_feasibility.png")
    if os.path.exists(kl_png):
        from source.visualization import compose_migration_vs_kl
        combo_png = os.path.join(args.outdir, "fig3_migration_vs_KL.png")
        compose_migration_vs_kl(disk_png, kl_png, combo_png)
        logger.info(f"Combined figure saved to {combo_png}")

# Update KozaiLidov analyzer with new parameters
if args.kozai_lidov:
    logger.info("Running Kozai-Lidov feasibility analysis...")

    kl_analyzer = KozaiLidovAnalyzer(
        data,
        toi_data=toi_data,
        inner_a0_AU=args.kl_a0,
        horizon_Gyr=args.kl_horizon_gyr,
        rpcrit_Rs=args.rpcrit_Rs
    )

    kl_results = kl_analyzer.analyze_feasibility()
    # ... rest of existing KL analysis ...
```

3.3 Add Visualization Compositor

File to modify: `source/visualization.py`

Add the following function to the `VLMSVisualizer` class or as a standalone function:

```python
def compose_migration_vs_kl(disk_png, kl_png, out_png):
    """
    Create side-by-side comparison of disk migration and KL feasibility

    Parameters:
    -----------
    disk_png : str
        Path to disk migration figure
    kl_png : str
        Path to KL feasibility figure
    out_png : str
        Output path for combined figure
    """
    import matplotlib.image as mpimg

    # Load images
    img_disk = mpimg.imread(disk_png)
    img_kl = mpimg.imread(kl_png)

    # Create composite figure
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6),
                                   constrained_layout=True)

    ax1.imshow(img_disk)
    ax1.axis('off')
    ax1.set_title("Disk Migration Timescales", fontsize=12)

    ax2.imshow(img_kl)
    ax2.axis('off')
    ax2.set_title("Kozai-Lidov + Tides Feasibility", fontsize=12)

    fig.suptitle("Inward Hardening Pathways for VLMS Companions",
                 fontsize=14, fontweight='bold')

    fig.savefig(out_png, dpi=200, bbox_inches='tight')
    plt.close(fig)
```

Phase 4: Testing and Validation

4.1 Unit Tests

New file to create: `tests/test_disk_migration.py`

```python
"""Tests for disk migration module"""
import pytest
import numpy as np
```

```python
from source.disk_migration import typeI_timescale_sec, migrate_time_numeric

def test_typeI_timescale():
    """Test Type-I migration timescale calculation"""
    # Test with typical VLMS parameters
    t_mig = typeI_timescale_sec(
        Mstar_Msun=0.1,
        Mp_Mj=0.3,
        a_AU=1.0,
        Sigma1_gcm2=300,
        p_sigma=1.0,
        H_over_a=0.04
    )

    # Should be on order of Myr
    t_mig_myr = t_mig / (1e6 * 365.25 * 24 * 3600)
    assert 0.1 < t_mig_myr < 100, f"Unexpected timescale: {t_mig_myr} Myr"

def test_migrate_time_numeric():
    """Test numerical integration of migration"""
    t_total = migrate_time_numeric(
        Mstar_Msun=0.1,
        Mp_Mj=0.3,
        a0_AU=1.0,
        af_AU=0.05,
        Sigma1_gcm2=300,
        p_sigma=1.0,
        H_over_a=0.04,
        nstep=100  # Fewer steps for test
    )

    t_total_myr = t_total / (1e6 * 365.25 * 24 * 3600)
    assert 0.1 < t_total_myr < 10, f"Unexpected total time: {t_total_myr} Myr"
```

New file to create: `tests/test_regime_clustering.py`

```python
"""Tests for regime discovery module"""
import pytest
import pandas as pd
import numpy as np
from source.analysis.regime_clustering import run_hdbscan_and_gmm

def test_hdbscan_clustering(tmp_path):
    """Test HDBSCAN clustering on synthetic data"""
    # Create synthetic system data
    np.random.seed(42)
    n = 100

    # Two distinct populations
    pop1_q = np.random.normal(-3, 0.3, n//2)  # Low q
    pop1_a = np.random.normal(-0.5, 0.2, n//2)  # Small a

    pop2_q = np.random.normal(-1.5, 0.3, n//2)  # High q
    pop2_a = np.random.normal(0.5, 0.2, n//2)  # Large a
```

```python
    df = pd.DataFrame({
        'host': [f"host_{i}" for i in range(n)],
        'logq_max': np.concatenate([pop1_q, pop2_q]),
        'loga_min': np.concatenate([pop1_a, pop2_a]),
        'e_max': np.random.uniform(0, 0.5, n)
    })

    test_csv = tmp_path / "test_systems.csv"
    df.to_csv(test_csv, index=False)

    # Run clustering
    result_df, bic_info = run_hdbscan_and_gmm(
        systems_csv=str(test_csv),
        out_prefix=str(tmp_path / "test_regimes"),
        min_cluster_size=10
    )

    # Should find multiple clusters
    n_clusters = len(result_df['hdbscan_label'].unique())
    assert n_clusters >= 2, f"Expected at least 2 clusters, got {n_clusters}"

    # GMM should prefer 2 components for this synthetic data
    assert bic_info['K_best'] == 2, f"Expected K=2, got K={bic_info['K_best']}"
```

4.2 Integration Tests

File to modify: `tests/test_command_line_integration.py`

Add new test cases:

```python
def test_disk_panel_integration(sample_data_dir):
    """Test disk migration panel generation"""
    result = subprocess.run([
        'python', 'source/panoptic_vlms_project.py',
        '--local-file', str(sample_data_dir / 'test_data.csv'),
        '--disk-panel',
        '--disk-lifetime-myr', '3.0',
        '--Sigma1AU', '300',
        '--outdir', 'test_output'
    ], capture_output=True, text=True)

    assert result.returncode == 0
    assert os.path.exists('test_output/fig3_disk.png')

def test_regime_discovery_integration(sample_data_dir):
    """Test regime discovery pipeline"""
    # First build systems
    result = subprocess.run([
        'python', 'source/panoptic_vlms_project.py',
        '--local-file', str(sample_data_dir / 'test_data.csv'),
        '--build-systems',
        '--regimes',
```

```python
        '--outdir', 'test_output'
    ], capture_output=True, text=True)

    assert result.returncode == 0
    assert os.path.exists('test_output/regimes_labels.csv')
    assert os.path.exists('test_output/segmented_logq_loga.png')
```

Phase 5: Documentation and Usage

5.1 Update README

Add the following section to `README.md`:

```
## New Features (v2.0)

### Disk Migration Analysis
Calculate Type-I migration timescales to assess the feasibility of disk-driven inward migration:

```bash
python source/panoptic_vlms_project.py \
    --fetch \
    --disk-panel \
    --disk-lifetime-myr 3.0 \
    --Sigma1AU 300 \
    --H-over-a 0.04
```

Statistical Regime Discovery

Identify distinct companion populations using clustering and segmented regression:

```
python source/panoptic_vlms_project.py \
    --fetch \
    --build-systems \
    --regimes
```

Enhanced Kozai-Lidov Analysis

Updated secular dynamics with customizable birth radii and time horizons:

```
python source/panoptic_vlms_project.py \
    --fetch \
    --kozai-lidov \
    --kl-a0 0.5 \
    --kl-horizon-gyr 3.0 \
    --rpcrit-Rs 3.0
```
```

System-Level Analysis with Stellar Binaries

Include VLMS stellar binaries in the analysis:

````
# Prepare stellar binary catalog with columns:
# host, Mstar_Msun, M2_Msun, a_AU, e, Age_Gyr, FeH

python source/panoptic_vlms_project.py \
    --fetch \
    --build-systems \
    --sb-csv data/vlms_binaries.csv \
    --regimes




### 5.2 Create Enhancement Test Script

**New file to create:** `test_enhancements.py`

```python
#!/usr/bin/env python
"""
Test script to verify all enhancements are working correctly
"""
import subprocess
import os
import sys

def test_enhancement(cmd, expected_files, description):
    """Run a test and check outputs"""
    print(f"\nTesting: {description}")
    print(f"Command: {' '.join(cmd)}")

    result = subprocess.run(cmd, capture_output=True, text=True)

    if result.returncode != 0:
        print(f"❌ Failed with error:\n{result.stderr}")
        return False

    missing = []
    for f in expected_files:
        if not os.path.exists(f):
            missing.append(f)

    if missing:
        print(f"❌ Missing expected files: {missing}")
        return False

    print("✅ Success!")
    return True

def main():
    """Run all enhancement tests"""
    tests = [
````
````

```
        {
            'cmd': ['python', 'source/panoptic_vlms_project.py',
                   '--fetch', '--percentage', '10'],
            'files': ['results/pscomppars_lowM.csv'],
            'desc': 'Basic data fetch'
        },
        {
            'cmd': ['python', 'source/panoptic_vlms_project.py',
                   '--local-file', 'results/pscomppars_lowM.csv',
                   '--build-systems'],
            'files': ['results/combined_systems.csv'],
            'desc': 'System table building'
        },
        {
            'cmd': ['python', 'source/panoptic_vlms_project.py',
                   '--local-file', 'results/pscomppars_lowM.csv',
                   '--build-systems', '--regimes'],
            'files': ['results/regimes_labels.csv',
                     'results/regimes_hdbscan_logq_loga.png'],
            'desc': 'Regime discovery'
        },
        {
            'cmd': ['python', 'source/panoptic_vlms_project.py',
                   '--local-file', 'results/pscomppars_lowM.csv',
                   '--disk-panel'],
            'files': ['results/fig3_disk.png'],
            'desc': 'Disk migration panel'
        }
    ]

    print("="*60)
    print("VLMS Enhancement Test Suite")
    print("="*60)

    passed = 0
    for test in tests:
        if test_enhancement(test['cmd'], test['files'], test['desc']):
            passed += 1

    print("\n" + "="*60)
    print(f"Results: {passed}/{len(tests)} tests passed")

    if passed == len(tests):
        print("🎉 All enhancements working correctly!")
        return 0
    else:
        print("⚠️  Some tests failed. Check the output above.")
        return 1

if __name__ == "__main__":
    sys.exit(main())
```

Phase 6: Deployment Checklist

Installation Steps

1. **Update dependencies:**

```
pip install -r requirements.txt
```

2. **Run tests to verify installation:**

```
pytest tests/test_disk_migration.py
pytest tests/test_regime_clustering.py
python test_enhancements.py
```

3. **Run example analysis:**

```
# Complete enhanced analysis pipeline
python source/panoptic_vlms_project.py \
    --fetch \
    --percentage 100 \
    --build-systems \
    --regimes \
    --disk-panel \
    --kozai-lidov \
    --kl-a0 0.5 \
    --kl-horizon-gyr 3.0 \
    --outdir results/enhanced_run
```

Expected Outputs

After successful implementation, you should have:

New data products:

- `results/combined_systems.csv` - System-level aggregated data
- `results/regimes_labels.csv` - Cluster assignments
- `results/msr_responsibilities.csv` - Regime probabilities

New visualizations:

- `results/fig3_disk.png` - Disk migration timescales
- `results/fig3_migration_vs_KL.png` - Combined pathways comparison
- `results/regimes_hdbscan_logq_loga.png` - Regime clusters
- `results/segmented_logq_loga.png` - Phase-shift detection

New analysis outputs:

- `results/segmented_logq_loga.json` - Break point statistics

- `results/regimes_gmm_bic.png` - Model selection curve

Backward Compatibility

All enhancements are designed to be optional and backward compatible:

- Existing command-line usage remains unchanged
- New features are activated only with specific flags
- Default behavior preserves original functionality
- All existing tests should continue to pass

Troubleshooting Guide

Common Issues and Solutions

1. **ImportError for hdbscan or ruptures:**

```
pip install hdbscan ruptures
```

2. **Memory issues with large datasets:**
   - Use `--percentage` flag to work with subsets
   - Reduce `nstep` in migration calculations
   - Use fewer grid points in feasibility maps
3. **Plotting issues:**
   - Ensure matplotlib backend is set correctly
   - Use `--no-display` flag for headless environments
4. **Numerical convergence warnings:**
   - Increase `max_iter` in EM algorithms
   - Check for data quality issues (NaNs, outliers)

This implementation guide provides a complete roadmap for integrating all enhancements while maintaining the integrity and functionality of the existing VLMS companion analysis system.