Experimental Projects Course

Weizmann Institute of Science

What is a Horizon?

A Fibre Optics Perspective

Eren Erberk Erkul

14 March 2026

Alles Gute zum Geburtstag, Herr Einstein!

Abstract.

This work contains the simulations and theoretical analysis I carried out during the semester break for the Experimental Projects course in the MSc curriculum at the Weizmann Institute of Science, for the fiber-optics tabletop Hawking-radiation experiment directed by Prof. Ulf Leonhardt.

Acknowledgment.

I am grateful to Mattan Gelvan for providing the initial simulation code, his clear explanations, and his enthusiasm. I also wish to thank my mentor, Ulf Leonhardt, who originated the ideas presented here and holds the copyright to this document.

Contents

1 Theoretical

Background

The concept of a horizon in Einstein's relativity is a paradoxical concept. It appears in an interrelated class of divergencies, namely in the boundary of black holes as a solution of the zero of the lapse function of a given black-hole metric, in the Hawking–Hartle cosmological horizon, and from the Unruh effect. Although the spontaneous particle creation shared between these horizons is suggestive, there still appears to be no complete generalisation of the whole class of horizons under the same mathematical structure. Moreover, we still lack the theory of Quantum Gravity, and these spontaneous particle creations are derived using nonrigorous semiclassical techniques. Hence, in the literature, it is widely accepted that these calculations are clues for a greater theory that still awaits us ever since Hawking's initial work [5]. So, the concept of gravitational horizon is not yet fully established and generalised.

So in this project, the aim was to explore what we can learn about the concept of gravitational horizon from its optical counterpart, and what it may teach us about the general properties of such spontaneous particle creation. To do this, one needs a strong dictionary between the two domains in order to translate back and forth the mathematical structure. It is already established that general relativity and classical electrodynamics have a strong mathematical commonality in their structure. Hence, in this report, we will take as given that the relevant optical solutions have a direct counterpart in the horizon kinematics of general relativity, even though this is not obvious at first [2, 6].

Mathematically, we follow the Hamiltonian formulation of the optical analogue developed by Prof. Ulf Leonhardt. Briefly, the idea is the following. One describes the pump and probe fields by amplitudes A_1 and A_2, whose analytic-signal components are a_1 and a_2. In the co-moving frame the evolution is generated by a Hamiltonian

$$H = H_1 + H_2 + H_{\text{int}}, \tag{1.1}$$

where H_1 and H_2 are the free-propagation Hamiltonians of the pump and probe, respectively, and H_{int} is the Kerr interaction Hamiltonian of the form

$$H_{\text{int}} = 16\kappa A_1^2 A_2^2. \tag{1.2}$$

When expanded in terms of a_m and a_m^*, this interaction contains several elementary processes, but the most important ones here are

$$H_S = 4\kappa\, a_1^* a_1 a_2^* a_2, \qquad H_R = 2\kappa\, a_1^* a_1 \left(a_2^{*2} + a_2^2\right). \tag{1.3}$$

The first term H_S gives the refractive-index shift produced by the pump, namely the cross-phase modulation that creates the effective horizon. The second term H_R is the pair-creation term: it is the part of the interaction that generates the Hawking partner channel and, correspondingly, the backreaction on the pump. Thus both ingredients are needed: H_S establishes the moving medium, while H_R is the actual radiation-generating process.

The pump pulse induces, through the Kerr effect, a refractive-index perturbation

$$n(\omega, \tau) = n_0(\omega) + \delta n(\tau), \tag{1.4}$$

where $\tau = t - z/u$ is the retarded time in the frame of the pump moving with velocity u. The local

propagation constant is then

$$\beta(\omega, \tau) = \frac{n(\omega, \tau)\,\omega}{c}. \tag{1.5}$$

In the co-moving frame the quantity

$$\omega' = \omega - u\,\beta(\omega, \tau) \tag{1.6}$$

plays the role of a conserved Hamiltonian. Equivalently one may write

$$\mathcal{H}(\tau, \omega) = \omega - u\,\beta(\omega, \tau). \tag{1.7}$$

Equation (1.7) acts as the effective co-moving ray Hamiltonian generating the phase-space dynamics in (τ, ω). This is the central quantity of the optical horizon picture, because the branch structure in the moving frame is generated by it.

The corresponding Hamilton equations are

$$\frac{d\tau}{dz} = -\frac{\partial \mathcal{H}}{\partial \omega} = -1 + u\,\frac{\partial \beta}{\partial \omega}, \qquad \frac{d\omega}{dz} = \frac{\partial \mathcal{H}}{\partial \tau} = -u\,\frac{\partial \beta}{\partial \tau}. \tag{1.8}$$

The first equation determines the drift of a mode relative to the moving pump profile, while the second equation describes how the mode's frequency changes in a nonuniform Kerr background. Since

$$\frac{\partial \beta}{\partial \omega} = \frac{1}{v_g}, \tag{1.9}$$

the first Hamilton equation in Eq. (1.8) becomes

$$\frac{d\tau}{dz} = -1 + \frac{u}{v_g}. \tag{1.10}$$

Therefore the turning-point or horizon condition is simply

$$\frac{d\tau}{dz} = 0 \qquad \Longleftrightarrow \qquad v_g = u. \tag{1.11}$$

So the optical horizon is the point where the group velocity of the probe relative to the medium equals the velocity of the pump pulse. At that point the probe can no longer escape across the moving Kerr profile. More precisely, the relevant group velocity is the one in the Kerr-shifted medium established by the pump, so the horizon is created by H_S while the actual Hawking pair generation is governed by H_R.

As shown in Figure 1, when the probe pulse approaches the trailing edge of the pump it gets stuck there akin to a black-hole horizon.

The same statement may be written directly as a constant-ω' scattering problem:

$$\omega - u\,\beta(\omega, \tau) = \omega'_0. \tag{1.12}$$

For a fixed value of ω'_0, Eq. (1.12) can admit multiple laboratory-frequency roots ω. In that sense, the horizon problem is naturally a mode-conversion problem between different branches that share the same co-moving invariant. This is exactly the mathematical structure that later appears in the numerical code when we solve for the NHR (Negative Hawking Radiation) and DRR (Double Resonant Radiation) branches from a fixed ω'.

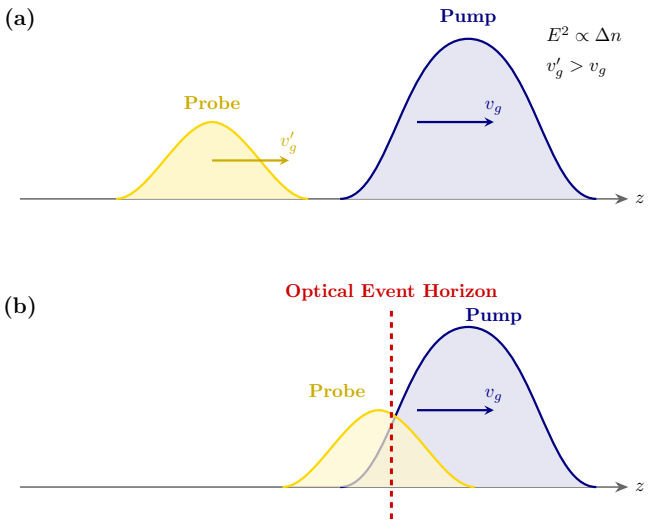

Figure 1: Kinematic representation of the fibre-optical event horizon. **(a)** A faster probe pulse catches up to a slower pump pulse. **(b)** The probe pulse coincides with the trailing edge of the pump. The Kerr-induced refractive-index change creates an effective horizon where the probe can no longer advance.

Near the horizon, one expands the Hamiltonian around a turning point (τ_h, ω_h) satisfying

$$\mathcal{H}(\tau_h, \omega_h) = \omega_0', \qquad \left.\frac{\partial \mathcal{H}}{\partial \omega}\right|_{(\tau_h, \omega_h)} = 0. \tag{1.13}$$

Then, to the lowest nontrivial order, one gets

$$\mathcal{H} - \omega_0' \approx \frac{1}{2}\mathcal{H}_{\omega\omega}\,(\omega - \omega_h)^2 + \mathcal{H}_\tau\,(\tau - \tau_h), \tag{1.14}$$

which is the local normal form of a turning-point Hamiltonian. Equation (1.14) is the origin of the familiar Airy-type scattering behaviour and is the mathematical reason the horizon acts as a converter between different asymptotic branches. In other words, the optical Hawking problem is not mysterious at the formal level, it is a dispersive Hamiltonian scattering problem with a moving inhomogeneous background.

Hence, in the experiment, one launches a weak probe on one asymptotic branch and lets the moving Kerr profile scatter it into the other allowed roots with the same co-moving invariant ω'. The ultraviolet channels that are later labeled NHR and DRR are therefore not inserted by hand, but arise intrinsically from the branch structure of the Hamiltonian relation. At the same time, from the Hamiltonian point of view, the existence of these partner channels is not only a kinematic statement about the branches, but also a dynamical one where the horizon-forming term H_S and the radiation-generating term H_R act together, the former establishing the effective moving background and the latter producing the Hawking pair and its backreaction. Hence in this project our aim is to analyze whether the UV spectrum directly resembles that of Hawking radiation.

Theoretical Speculations

Here, our aim is not to rederive the optical horizon itself, but to ask whether the mathematical structure identified above may be more general. In particular, we ask whether Hawking-like spectral leakage may arise in a broader class of systems governed by suitable partial differential equations with moving inhomogeneous backgrounds and nontrivial branch structure. This is precisely why the fibre-optical system is such a useful laboratory model as it isolates the mathematical core of the horizon process without requiring the full machinery of quantum gravity.

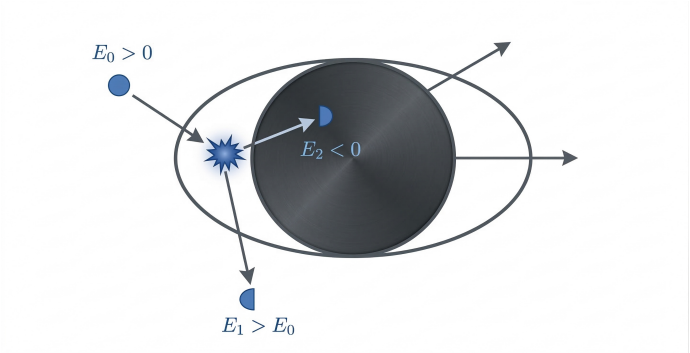

Figure 2: Schematic of the Penrose process. An incident particle with initial energy $E_0 > 0$ enters the ergosphere and splits. One fragment acquires negative energy $E_2 < 0$ and falls past the event horizon. The remaining fragment escapes to infinity with amplified energy $E_1 > E_0$, thereby extracting rotational energy from the black hole.

The Penrose process is a mechanism for extracting energy from a rotating Kerr black hole. Roughly speaking, radiation or matter falling towards a black hole is blueshifted with respect to an observer at infinity, which already suggests that a rotating black hole can in principle act as an energy source. Penrose therefore proposed a gedanken experiment in which a particle enters the ergosphere and splits into two parts: one falls through the horizon with negative energy relative to infinity, while the other escapes with greater energy than the original incident particle, so that energy is extracted from the black hole's rotation while the total energy remains conserved.

Inspired by this idea, Yakov Zel'dovich generalised the mechanism to classical rotating dissipative systems beyond the Kerr geometry [3, 4]. Shortly afterwards, his student Starobinsky extended the analysis further and showed that analogous amplification phenomena could arise for wave scattering in rotating backgrounds more generally. Zel'dovich also argued that quantum fluctuations in such systems should lead to spontaneous emission. This line of thought played an important conceptual role in the development of Hawking's later work on black-hole radiation [5, 9].

Hence, one should attempt to approach the question in the spirit of Zel'dovich. Hawking radiation may well reflect a more general kinematical structure already present in certain classical systems that exhibit superradiant or horizon-like behaviour, with quantum theory determining the spontaneous character of the emission. It is then natural to ask whether other dynamical systems, equipped with the appropriate boundary conditions and branch structure, may also produce Hawking-like radiation.

Hence, in the optical system, the moving Kerr profile provides a classical Hamiltonian scattering problem whose branch conversion imitates the kinematical structure of horizon physics.

The proposed idea, then, is that the gravitational setting and the fibre-optical setting share the same essential mathematical structure: the gravitational well is mirrored by the optical background, while

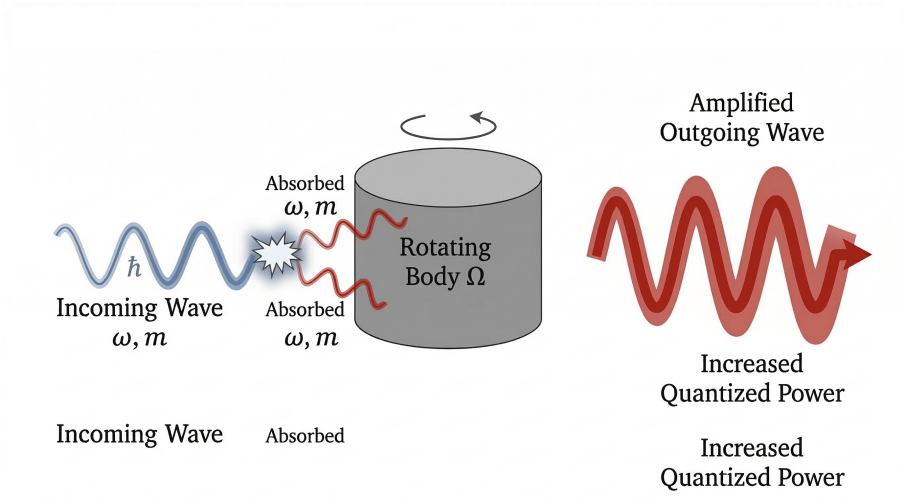

Figure 3: Schematic of Zel'dovich superradiance. A wave incident on a rotating absorbing body can undergo amplification under the condition $\omega < m\Omega$, illustrating the classical rotational energy-extraction mechanism that later became central to the broader discussion of Hawking-like emission.

the moving Kerr perturbation plays the role of the travelling inhomogeneity that drives redshift, mode conversion, and branch scattering. In this sense, there is a near one-to-one correspondence between the mathematical problem underlying Penrose-type horizon physics and that of the fibre-optics experiment proposed here. What the numerics test later is precisely whether the stimulated ultraviolet output tracks those Hamiltonian branches in the manner predicted by the theory. However, due to the constraints of this course's time frame, we will delay the theoretical conjecture outlined here to a later time.

Model, notation, and numerical grids

The simulation is formulated as a one-dimensional propagation problem. The field evolves along the fibre coordinate z, while its temporal structure is resolved on a finite retarded-time window t. Numerically, we work with the analytic signal $a(t, z)$ rather than the real field itself, so that only positive laboratory frequencies are propagated explicitly. This is the natural representation for the code, since dispersion acts diagonally in frequency space while the Kerr response is evaluated in time.

Units

Internally we use t in femtoseconds, z in micrometers, ω in radians per femtosecond, and c in micrometers per femtosecond. With this convention, wavelengths are naturally expressed in micrometers and the conversion

$$\omega = \frac{2\pi c}{\lambda} \tag{1.15}$$

is used directly throughout the implementation.

FFT grids

With N_t samples over a time window T, the discrete time and frequency grids are

$$\Delta t = \frac{T}{N_t}, \qquad t_n = \left(n - \frac{N_t}{2}\right)\Delta t, \qquad \omega_k = 2\pi\,\mathrm{fftfreq}(N_t, \Delta t). \tag{1.16}$$

The code uses the FFT pair associated with the discretization in Eq. (1.16). The linear dispersive step is therefore applied directly on the ω_k grid, whereas the nonlinear response is evaluated in the time domain and then transformed back to frequency space. The positive-frequency condition of the analytic signal is enforced by projection on the same ω_k grid.

Governing equation

The propagated spectral field

$$A(\omega, z) = \mathcal{F}[a(t, z)] \tag{1.17}$$

obeys an analytic-signal UPPE-type equation of the form

$$\partial_z A(\omega, z) = -\mathrm{i}\,\beta_{\mathrm{eff}}(\omega)\,A(\omega, z) - \mathrm{i}\,\theta(\omega)\left(\mathcal{F}\{\mathcal{N}[a]\}\right)_+, \tag{1.18}$$

where $(\cdot)_+ \equiv \mathcal{P}_+$ denotes projection onto positive laboratory frequencies. Here $\beta_{\mathrm{eff}}(\omega)$ is either the laboratory propagation constant $\beta(\omega)$ or, when the co-moving option is used, the shifted quantity $\beta(\omega) - \omega/u$. The factor $\theta(\omega)$ is the frequency-dependent Kerr prefactor in the normalization adopted by the code.

The nonlinear term is implemented as

$$\mathcal{N}[a] = a^3 + 3|a|^2 a + 3|a|^2 a^*. \tag{1.19}$$

This is the form actually used by the numerical solver and is fully consistent with the theory described in previous section. In practice, the code alternates between frequency space and time space implying the dispersive part is updated spectrally, while the nonlinear source is constructed in time and returned to frequency by FFT. In the stimulated runs, the evolution law in Eq. (1.18) is solved once for the pump alone and once for the pump plus probe, and the ultraviolet signal of interest is then extracted from their difference.

2 Simulation

config.py

Wavelength and frequency conversion

```python
def omega0(self, lambda_um: float) -> float:
    return 2.0 * np.pi * self.c / float(lambda_um)

def lambda_from_omega(self, omega):
    omega = np.asarray(omega, dtype=float)
    lam = np.full_like(omega, np.inf, dtype=float)
    m = omega != 0.0
    lam[m] = 2.0 * np.pi * self.c / omega[m]
    return lam
```

Code 1: config.py: omega0 and lambda conversion

This establishes the dictionary between the laboratory language of wavelengths and the numerical language of angular frequencies.

$$\omega(\lambda) = \frac{2\pi c}{\lambda}. \tag{2.1}$$

The $\omega = 0$ bin is mapped to $\lambda = \infty$, which prevents division by zero while ensuring no finite-ω physics is altered.

Pump and probe relative amplitudes

```python
def pump_probe_amplitudes(self) -> tuple[float, float]:
    P1 = float(self.P_pump_mW) * float(self.pump_coupling_eff)
    P2 = float(self.P_probe_mW)
    if P1 <= 0:
        return 1.0, 0.0
    ratio = np.sqrt(max(P2, 0.0) / P1)
    return 1.0, float(ratio)
```

Code 2: config.py: pump_probe_amplitudes

This sets the relative scaling between the two input envelopes. The pump amplitude is fixed to 1 in the internal units of the repository, and the probe amplitude is set by a square-root power ratio. There is no SI calibration step here; this is a controlled way to ensure that the probe is weaker than the pump with the same relative hierarchy as the experiment.

Grid sizes and step counts

```python
def dt(self) -> float:
    return float(self.T_window) / float(self.Nt)

def zsteps(self) -> int:
    if self.dz <= 0:
        return 1
    return int(np.round(self.z_total / self.dz))
```

Code 3: config.py: dt and zsteps

Propagation is explicit in z, so the step size Δz sets accuracy and cost. Time is resolved by FFT, so $\Delta t = T/N_t$ sets spectral resolution and bandwidth. In practice, T must contain the full nonlinear interaction without wrap-around, and N_t must support the UV features without aliasing.

Pulse width conversion for sech profiles

```python
def fwhm_to_T0_sech(self, fwhm_fs: float) -> float:
    return float(fwhm_fs) / 1.763
```

Code 4: config.py: FWHM to T0 conversion

For $I(t) = \mathrm{sech}^2(t/T_0)$, the intensity full-width at half-maximum is FWHM $\approx 1.763\,T_0$, hence $T_0 =$ FWHM$/1.763$. This is what controls the bandwidth of the carrier-modulated envelope in a reproducible way.

Dispersion model and propagation constant

```python
def n(self, omega):
    omega = np.asarray(omega, dtype=float)
    if self.n_func is not None:
        return self.n_func(omega)
    denom = (self.w_res**2 - omega**2)
    denom = np.where(np.abs(denom) < 1e-6*self.w_res**2,
                     np.sign(denom)*1e-6*self.w_res**2, denom)
    return self.n0 + self.chi/denom

def beta(self, omega):
    omega = np.asarray(omega, dtype=float)
    if self.truncate_beta:
        return 0.5 * self.beta2 * omega**2
    return self.n(omega) * omega / self.c
```

Code 5: config.py: refractive index and beta

The main architecture uses

$$\beta(\omega) = \frac{n(\omega)\,\omega}{c}. \tag{2.2}$$

In the present implementation, the simplest complete effective profile is taken to be

$$n(\omega) = n_0 + \frac{\chi}{\omega_{\mathrm{res}}^2 - \omega^2}, \tag{2.3}$$

where n_0 is the background refractive index, χ sets the strength of the dispersive correction, and ω_{res} is the effective resonance frequency of the model. This is a simplified Lorentz-type resonance profile, chosen as the minimal dispersive model capable of producing a nontrivial ultraviolet branch structure and a resonant behaviour in the spectrum. It should therefore be understood as an effective profile rather than as a fully realistic fit to the experimental fibre, and in future iterations it should be replaced by a more accurate experimentally calibrated dispersion model.

The quadratic truncation exists only for the legacy demo (Mattan's first version) and is not the dispersion used in the Hawking/backreaction runs. The clipping around the model pole is a numerical regularization of the effective index, introduced only to avoid divergence when ω approaches ω_{res}.

Pump velocity and co-moving frequency

```python
def u_pump(self) -> float:
    return self.c / float(self.n_g)

def omega_comoving(self, omega):
    omega = np.asarray(omega, dtype=float)
    u = self.u_pump()
    return (1.0 - self.n(omega) * u / self.c) * omega
```

Code 6: config.py: pump speed and co-moving frequency

This implements the co-moving branch label introduced in the theoretical section:

$$\omega'(\omega) = \omega - u\,\beta(\omega) = \Big(1 - \frac{n(\omega)u}{c}\Big)\omega, \qquad u = \frac{c}{n_g}. \tag{2.4}$$

All theory targets in ω' are mapped back to laboratory ω by solving Eq. (2.4) for ω.

Kerr prefactor

```python
def kerr_theta(self, omega):
    omega = np.asarray(omega, dtype=float)
    n = self.n(omega)
    theta = omega * (self.chi3 * self.chi3_scale) / (16.0 * self.c * n)
    theta = np.where(omega == 0.0, 0.0, theta)
    return theta
```

Code 7: config.py: Kerr theta

The solver writes the nonlinear source as a spectrum multiplied by $\theta(\omega)$. In the analytic-signal conventions used here,

$$\theta(\omega) = \frac{\omega\,\chi^{(3)}}{16\,c\,n(\omega)}. \tag{2.5}$$

The scale factor `chi3_scale` is the knob that sets nonlinear strength in this repository's internal units.

Configuration presets: fast demo vs paper-like

```python
def paper_config(fast: bool = True) -> Config:
    cfg = Config()
    cfg.truncate_beta = False
    cfg.n_func = cfg.generate_n_func()
    cfg.lambda_pump = 0.800
    cfg.lambda_probe = 1.400
    cfg.fwhm_pump = 8.0
    cfg.fwhm_probe = 30.0
    cfg.z_total = 7000.0
    cfg.n_g = float(cfg.n(cfg.omega0(cfg.lambda_pump)) + 5e-6)
    cfg.use_comoving_frame = True
    cfg.enforce_analytic = True
    if fast:
        cfg.Nt = 2**13
        cfg.T_window = 1600.0
        cfg.dz = 10.0
    else:
        cfg.Nt = 2**15
        cfg.T_window = 3176.0
```

```
        cfg.dz = 0.5
    return cfg

def default_config() -> Config:
    return paper_config(fast=True)
```

Code 8: config.py: paper_config and default_config

The presets control whether one runs a fast demonstration or a heavier run closer to the paper's numerical section. The key toggles are the grid size N_t, the time window T, and the propagation step dz. In both cases, the same dispersion and the same co-moving mapping are used, so the comparison logic remains consistent. However, due to the limitations of my computer or other reasons that I am not aware of, my computer couldn't produce a consistent result for the fourth plot, hence the results are based on the fast configuration. In the conclusion, I contemplate this point in greater detail.

solver.py

Time and frequency grids

```
def make_time_grid(cfg):
    dt = cfg.dt()
    return (np.arange(cfg.Nt) - cfg.Nt//2) * dt

def make_omega_grid(t):
    dt = float(t[1] - t[0])
    return 2.0*np.pi*np.fft.fftfreq(t.size, d=dt)
```

Code 9: solver.py: time grid and omega grid

The time grid is centred so delays are symmetric and FFT phases behave cleanly. The frequency grid is the exact companion of that discretization. The solver uses the FFT bins implied by t for both dispersion and projection.

Positive-frequency projector

```
def project_positive(A_w, omega):
    out = np.zeros_like(A_w)
    out[omega >= 0.0] = A_w[omega >= 0.0]
    return out
```

Code 10: solver.py: positive-frequency projection

This implements

$$(\mathcal{P}_+ A)(\omega) = \Theta(\omega) A(\omega). \tag{2.6}$$

In the analytic-signal formulation this is the enforcement that the propagated field contains only positive laboratory frequencies, while the conjugate contribution is carried explicitly by the nonlinear term.

Analytic-signal Kerr functional

```python
def kerr_nonlinearity_time(a_t):
    abs2 = np.abs(a_t)**2
    return a_t**3 + 3.0*abs2*a_t + 3.0*abs2*np.conj(a_t)
```

Code 11: solver.py: Kerr nonlinearity in time

This is the full cubic structure used by the Hawking/backreaction code:

$$\mathcal{N}[a] = a^3 + 3|a|^2 a + 3|a|^2 a^*. \tag{2.7}$$

These terms are the computational channels for four-wave mixing between pump and probe bands, including conjugate mixing required by the analytic-signal convention.

Spectral right-hand side

```python
def rhs_uppe(A_w, omega, beta_eff, theta, cfg):
    a_t  = np.fft.ifft(A_w)
    NL_t = kerr_nonlinearity_time(a_t)
    NL_w = np.fft.fft(NL_t)
    if cfg.enforce_analytic:
        NL_w = project_positive(NL_w, omega)
    return -1j*(beta_eff*A_w + theta*NL_w)
```

Code 12: solver.py: spectral right-hand side

This is the direct discretization of

$$\partial_z A = -\mathrm{i}\,\beta_{\mathrm{eff}}(\omega)\,A - \mathrm{i}\,\theta(\omega)\left(\mathcal{F}\{\mathcal{N}[a]\}\right)_+. \tag{2.8}$$

The order "IFFT → build $\mathcal{N}$ → FFT → project" is the numerical implementation of the projected nonlinear source in Eq. (2.8).

Split-step propagation

```python
beta = cfg.beta(omega)
if cfg.use_comoving_frame:
    beta = beta - omega/cfg.u_pump()
theta = cfg.kerr_theta(omega)

lin_half = np.exp(-1j*beta*dz*0.5)
for _ in range(zsteps):
    A_w = A_w * lin_half
    a_t  = np.fft.ifft(A_w)
    NL_t = kerr_nonlinearity_time(a_t)
    NL_w = np.fft.fft(NL_t)
    if cfg.enforce_analytic:
        NL_w = project_positive(NL_w, omega)
    A_w = A_w + (-1j*dz)*(theta*NL_w)
    if cfg.enforce_analytic:
        A_w = project_positive(A_w, omega)
    A_w = A_w * lin_half
```

Code 13: solver.py: split-step propagation loop

The linear part is exactly solvable in frequency space as multiplication by $e^{-i\beta(\omega)\Delta z}$. The nonlinear part is local in time, so it is evaluated in t after IFFT. The co-moving option replaces $\beta(\omega)$ by $\beta(\omega) - \omega/u$, which is the spectral form of removing pump-frame phase advance.

Legacy wrapper: `propagate_Probe_Pulse`

```python
def propagate_Probe_Pulse(t, omega,
                          in0_t, out0_t, in0_w, out0_w,
                          cfg, d: float = 1.0, iteration_index: int = 0, u_t = None,
                          method: str = "Linear", return_time: bool = True):
    if method == "Linear":
        beta = cfg.beta(omega)
        phi = np.exp(1j * beta * d)
        in1_w = in0_w * phi
        out1_w = out0_w * phi
        if return_time:
            return in1_w, out1_w, np.fft.ifft(in1_w), np.fft.ifft(out1_w)
        return None
```

Code 14: solver.py: propagate_Probe_Pulse legacy wrapper

This code corresponds to the simplest linear propagation picture for the older reduced demo of Mattan. For a monochromatic component of angular frequency ω, the field evolves as

$$A(\omega, z) = A(\omega, 0)\, e^{i\beta(\omega)z}, \tag{2.9}$$

which solves

$$\frac{\partial A(\omega, z)}{\partial z} = i\,\beta(\omega)\, A(\omega, z). \tag{2.10}$$

So each spectral component propagates independently and acquires only a linear dispersive phase.

However, in the actual Hawking/backreaction runs this linear monochromatic evolution is no longer sufficient. There the code propagates the analytic signal with both the dispersive term and the nonlinear Kerr source, which is what allows mode conversion, four-wave mixing, and the appearance of the NHR and DRR channels.

main.py

Building the initial pump–probe field

```python
def sech(x):
    return 1.0 / np.cosh(x)

def build_initial_fields(cfg, t):
    pump_amp, probe_amp = cfg.pump_probe_amplitudes()
    T0_1 = cfg.fwhm_to_T0_sech(cfg.fwhm_pump)
    T0_2 = cfg.fwhm_to_T0_sech(cfg.fwhm_probe)
    w1 = cfg.omega0(cfg.lambda_pump)
    w2 = cfg.omega0(cfg.lambda_probe)
    env1 = pump_amp * sech((t - cfg.pump_delay) / T0_1)
    env2 = probe_amp * sech((t - cfg.probe_delay) / T0_2)
    a1 = env1 * np.exp(1j * w1 * t)
    a2 = env2 * np.exp(1j * w2 * t)
    return {
        "a0_pump_t": a1,
        "a0_probe_t": a2,
```

```
        "a0_both_t": a1 + a2,
        "w_pump": w1,
        "w_probe": w2,
    }
```

Code 15: main.py: sech and initial-field construction

The input analytic signal is written as the sum of two carrier-modulated sech pulses,

$$a(t,0) = a_1(t) + a_2(t), \qquad a_j(t) = A_j \operatorname{sech}\left(\frac{t - t_j}{T_{0,j}}\right) e^{i\omega_j t}. \tag{2.11}$$

Here A_j is the internal amplitude, t_j is the delay, and $T_{0,j}$ is the sech width obtained from the chosen FWHM. The function returns the pump-only, probe-only, and combined fields, together with the carrier frequencies ω_1 and ω_2, so that the rest of the code uses exactly the same initial spectral labels as the propagated field.

From co-moving frequency to laboratory branches

```
def solve_lab_omega_from_comoving(cfg, omega_prime_target, omega_max, ngrid=20000):
    w = np.linspace(1e-6, float(omega_max), int(ngrid))
    f = cfg.omega_comoving(w) - float(omega_prime_target)
    jumps = np.where(np.diff(np.sign(f)) != 0)[0]
    roots = []
    for j in jumps:
        a, b = w[j], w[j + 1]
        root = brentq(lambda ww: float(cfg.omega_comoving(ww) - omega_prime_target), a, b)
        roots.append(float(root))
    return sorted(roots)
```

Code 16: main.py: solving $\omega'(\omega) = \omega'_\star$

The theory is organized in terms of the co-moving conserved label introduced previously. For a fixed target value $\omega'_\star$, the code solves

$$\omega'(\omega) = \omega'_\star \tag{2.12}$$

for the corresponding laboratory frequency ω. Because dispersion can make $\omega'(\omega)$ non-monotone, one co-moving target can correspond to several laboratory roots. Numerically, the code samples $f(\omega) = \omega'(\omega) - \omega'_\star$, detects sign changes, and then applies Brent's method (is essentially a root finding algorithm that combines multiple methods) inside each bracket. The output is therefore not a single branch but a list of admissible laboratory-frequency branches.

Selecting the UV branch used in the comparison plots

```
def choose_root_in_window(cfg, roots, lam_min=0.18, lam_max=0.50):
    candidates = []
    for w in roots:
        lam = float(cfg.lambda_from_omega(w))
        if np.isfinite(lam) and (lam_min <= lam <= lam_max):
            candidates.append((w, lam))
    if not candidates:
        return np.nan
    return max(w for w, _ in candidates)
```

Code 17: main.py: selecting the plotted branch

This extra selection step is important. Since the theory comparison is plotted in a UV wavelength window, the code does not use an arbitrary laboratory root. Instead, after finding all solutions of Eq. (2.12), it keeps the branch that lies inside the plotting window

$$\lambda_{\min} \le \lambda(\omega) \le \lambda_{\max}, \qquad \lambda(\omega) = \frac{2\pi c}{\omega}. \tag{2.13}$$

In the present implementation this is the 0.18–0.50 μm window. This prevents the sweep panel and the single-run UV panel from accidentally following different mathematical branches of the same co-moving relation.

Co-moving theory targets and their laboratory images

```python
def hawking_theory(cfg, lambda_probe_um):
    w1 = cfg.omega0(cfg.lambda_pump)
    w2 = cfg.omega0(lambda_probe_um)

    w1_p = float(cfg.omega_comoving(w1))
    w2_p = float(cfg.omega_comoving(w2))

    wN_p = -w2_p
    wB_p = w1_p - 2.0 * w2_p

    omega_max = np.pi / cfg.dt() * 0.98
    roots_N = solve_lab_omega_from_comoving(cfg, wN_p, omega_max)
    roots_B = solve_lab_omega_from_comoving(cfg, wB_p, omega_max)

    wN = choose_root_in_window(cfg, roots_N, lam_min=0.18, lam_max=0.50)
    wB = choose_root_in_window(cfg, roots_B, lam_min=0.18, lam_max=0.50)

    lamN = float(cfg.lambda_from_omega(wN)) if np.isfinite(wN) else np.nan
    lamB = float(cfg.lambda_from_omega(wB)) if np.isfinite(wB) else np.nan

    return {
        "w_pump": w1,
        "w_probe": w2,
        "wprime_pump": w1_p,
        "wprime_probe": w2_p,
        "wprime_NHR": wN_p,
        "wprime_DRR": wB_p,
        "omega_NHR": wN,
        "omega_DRR": wB,
        "lambda_NHR_um": lamN,
        "lambda_DRR_um": lamB,
        "c": cfg.c,
    }
```

Code 18: main.py: theory targets and UV branch selection

The theoretical NHR and DRR targets are first built in the co-moving frame:

$$\omega'_{\mathrm{NHR}} = -\omega'_2, \qquad \omega'_{\mathrm{DRR}} = \omega'_1 - 2\omega'_2. \tag{2.14}$$

These are then mapped back to laboratory frequencies by solving the Doppler relation for ω. The function returns both levels of description: the co-moving targets $\omega'_{\mathrm{NHR}}, \omega'_{\mathrm{DRR}}$, and the selected laboratory frequencies and wavelengths

$$\omega_{\mathrm{NHR}},\ \omega_{\mathrm{DRR}}, \qquad \lambda_{\mathrm{NHR}} = \frac{2\pi c}{\omega_{\mathrm{NHR}}}, \quad \lambda_{\mathrm{DRR}} = \frac{2\pi c}{\omega_{\mathrm{DRR}}}. \tag{2.15}$$

This makes the theory comparison concrete where the invariants are formed in the moving frame, but the final comparison is done in the laboratory spectrum where the simulation is actually measured.

Stimulated subtraction

```
A_pump_w = propagate_uppe(init["a0_pump_t"], cfg)["a1_w"]
A_both_w = propagate_uppe(init["a0_both_t"], cfg)["a1_w"]
S_stim = np.maximum(np.abs(A_both_w)**2 - np.abs(A_pump_w)**2, 0.0)
```

Code 19: main.py: stimulated UV signal

The plotted UV signal is not the raw output spectrum but the probe-induced excess,

$$S_{\mathrm{stim}}(\omega) = \max\Big(|A_{\mathrm{both}}(\omega)|^2 - |A_{\mathrm{pump}}(\omega)|^2, 0\Big). \tag{2.16}$$

This subtraction removes the background generated by the pump alone and isolates the spectral weight that appears only when the probe is present. The point of the max is not to reduce the actual components of the true signal, but simply to suppress small negative residuals produced by finite-grid subtraction and numerical noise.

Peak extraction near each theory branch

```
def extract_peak_near(omega, spectrum, omega0, width=0.25):
    if not np.isfinite(omega0):
        return np.nan
    mask = (omega > 0.0) & (omega > omega0 - width) & (omega < omega0 + width)
    if not np.any(mask):
        return np.nan
    j = np.argmax(spectrum[mask])
    return float(omega[mask][j])
```

Code 20: main.py: branch-local peak extraction

Once theory predicts a branch center $\omega_\star$, the simulation peak is defined as the local maximum of the stimulated spectrum in a neighborhood of that theory value:

$$\omega_{\mathrm{peak}} = \underset{|\omega-\omega_\star|<w}{\arg\max}\ S_{\mathrm{stim}}(\omega). \tag{2.17}$$

This is important in the sweep. The goal is not to find the global maximum of the entire UV spectrum, but to ask whether the simulated signal near the predicted NHR or DRR location actually follows the corresponding theoretical branch as the probe wavelength is varied.

Template construction and correlation metric

```
def gaussian_template(omega, centers, sigma=0.25):
    tmp = np.zeros_like(omega, dtype=float)
    for c in centers:
        if np.isfinite(c):
            tmp += np.exp(-0.5 * ((omega - c) / sigma) ** 2)
    return tmp

def spectral_correlation(sim, ref):
```

```
    sim = np.asarray(sim, dtype=float)
    ref = np.asarray(ref, dtype=float)
    m = np.isfinite(sim) & np.isfinite(ref)
    if np.sum(m) < 10:
        return np.nan
    a = sim[m] - np.mean(sim[m])
    b = ref[m] - np.mean(ref[m])
    den = np.sqrt(np.sum(a * a) * np.sum(b * b))
    if den == 0.0:
        return np.nan
    return float(np.sum(a * b) / den)
```

Code 21: main.py: theory template and correlations

The theoretical UV spectrum used for visual comparison is represented by a two-peak Gaussian template centered at the predicted NHR and DRR frequencies. If those centers are ω_{NHR} and ω_{DRR}, then the template has the form

$$T(\omega) = \exp\left[-\frac{(\omega-\omega_{\mathrm{NHR}})^2}{2\sigma^2}\right] + \exp\left[-\frac{(\omega-\omega_{\mathrm{DRR}})^2}{2\sigma^2}\right]. \tag{2.18}$$

Although the stimulated spectrum is not available in closed analytic form, the code represents it numerically as sampled values $S_i = S_{\mathrm{stim}}(\omega_i)$ on the FFT frequency grid.

$$\rho_{\mathrm{spec}} = \frac{\sum_i (S_i-\bar{S})(T_i-\bar{T})}{\sqrt{\left[\sum_i (S_i-\bar{S})^2\right]\left[\sum_i (T_i-\bar{T})^2\right]}}, \tag{2.19}$$

The scalar spectral correlation is defined by Eq. (2.19). It is the normalized inner product between the stimulated simulation spectrum and the theory template after subtracting the means. It is therefore a shape-comparison metric rather than an absolute-power comparison of a direct Hawking formula.

```
def time_xcorr(x, y, dt):
    x = np.asarray(x, dtype=float)
    y = np.asarray(y, dtype=float)
    x = x - np.mean(x)
    y = y - np.mean(y)
    n = x.size
    m = 2 * n
    X = np.fft.fft(x, n=m)
    Y = np.fft.fft(y, n=m)
    corr = np.fft.ifft(X * np.conj(Y)).real
    corr = np.concatenate((corr[-(n - 1):], corr[:n]))
    lags = np.arange(-(n - 1), n)
    delay = lags * dt
    if np.max(np.abs(corr)) > 0:
        corr_n = corr / np.max(np.abs(corr))
    else:
        corr_n = corr
    return {"delay_fs": delay, "corr": corr, "corr_norm": corr_n}

def spectral_xcorr(x, y, domega):
    out = time_xcorr(x, y, domega)
    return {
        "shift_radfs": out["delay_fs"],
        "corr_spec": out["corr"],
        "corr_spec_norm": out["corr_norm"],
    }
```

Code 22: main.py: time and spectral cross-correlation

The time-domain cross-correlation is computed after filtering the stimulated complex spectrum around the NHR and DRR bands, transforming each band back to time, and forming the corresponding intensities $I_{\mathrm{NHR}}(t_n)$ and $I_{\mathrm{DRR}}(t_n)$. In the code, the mean values are first removed,

$$\tilde{I}_{\mathrm{NHR}}(t_n) = I_{\mathrm{NHR}}(t_n) - \overline{I}_{\mathrm{NHR}}, \qquad \tilde{I}_{\mathrm{DRR}}(t_n) = I_{\mathrm{DRR}}(t_n) - \overline{I}_{\mathrm{DRR}},$$

and the discrete linear cross-correlation is then evaluated as

$$C_k = \sum_n \tilde{I}_{\mathrm{NHR}}(t_n)\, \tilde{I}_{\mathrm{DRR}}(t_{n+k}), \tag{2.20}$$

with delay axis

$$\Delta t_k = k\, \Delta t.$$

The implementation uses the standard FFT method with zero-padding to obtain the linear, rather than circular, correlation. Afterward the correlation is normalized by its maximum absolute value,

$$C_k^{(\mathrm{norm})} = \frac{C_k}{\max_j |C_j|},$$

so the result indicates whether the two selected UV channels are temporally localized in the same nonlinear event, independent of their absolute scale.

The spectral version is mathematically the same construction applied to arrays indexed by ω rather than t, so its horizontal axis is a spectral shift $\Delta\omega_k = k\, \Delta\omega$.

Single-run - One ring to rule them all!!

```python
def run_single(cfg):
    t = make_time_grid(cfg)
    omega = make_omega_grid(t)
    init = build_initial_fields(cfg, t)

    res_pump = propagate_uppe(init["a0_pump_t"], cfg)
    A_pump_w = res_pump["a1_w"]

    res_both = propagate_uppe(init["a0_both_t"], cfg)
    A_both_w = res_both["a1_w"]

    theory = hawking_theory(cfg, cfg.lambda_probe)
    template = gaussian_template(omega, [theory["omega_NHR"], theory["omega_DRR"]], sigma=0.25)
    theory["template_uv"] = template

    S_stim = np.maximum(np.abs(A_both_w) ** 2 - np.abs(A_pump_w) ** 2, 0.0)
    A_stim_w = A_both_w - A_pump_w
    if cfg.enforce_analytic:
        A_stim_w = project_positive(A_stim_w, omega)
```

Code 23: main.py: propagation and stimulated spectrum in a single run

The single-run code first builds the common time and frequency grids introduced above, then propagates two initial conditions through the same UPPE solver, namely the pump alone and the pump plus probe. The theory prediction is generated for the same probe wavelength as the propagated input, using the NHR

and DRR branch construction discussed previously. At that point the code has three key spectral objects:

$$A_{\mathrm{pump}}(\omega), \qquad A_{\mathrm{both}}(\omega), \qquad S_{\mathrm{stim}}(\omega). \tag{2.21}$$

Here $S_{\mathrm{stim}}(\omega)$ is precisely the stimulated difference spectrum already defined in Eq. (2.16). In addition, it constructs an approximate complex stimulated field

$$A_{\mathrm{stim}}(\omega) \approx A_{\mathrm{both}}(\omega) - A_{\mathrm{pump}}(\omega), \tag{2.22}$$

which is then used for bandpass filtering and time-domain correlation analysis. This subtraction is only an operational metric; the physically plotted spectrum remains S_{stim} from Eq. (2.16). A small drawback is that $A_{\mathrm{stim}}(\omega)$ is not itself a directly measured observable, but rather a convenient constructed field used for the analysis which is also done in the analysis of Prof. Leonhardt.

```
    fN = np.exp(-0.5 * ((omega - theory["omega_NHR"]) / 0.40) ** 2)
    fB = np.exp(-0.5 * ((omega - theory["omega_DRR"]) / 0.40) ** 2)

    aN_t = np.fft.ifft(A_stim_w * fN)
    aB_t = np.fft.ifft(A_stim_w * fB)

    IN = np.abs(aN_t) ** 2
    IB = np.abs(aB_t) ** 2

    corr = time_xcorr(IN, IB, cfg.dt())
```

Code 24: main.py: bandpass filters and time-domain correlation

The two Gaussian filters $f_N(\omega)$ and $f_B(\omega)$ isolate narrow neighborhoods around the predicted NHR and DRR bands, in the same spirit as the two-peak template of Eq. (2.18). After inverse Fourier transform, the code obtains two band-limited time-domain signals whose intensities are then cross-correlated according to the discrete construction described in Eq. (2.20). In other words, the code asks whether the two theoretically selected UV channels are generated with similar temporal localization. A second small drawback is that the Gaussian windows although common in stochastic analysis are choices rather than unique physical objects, so their widths affect the output.

```
    result = {
        "t": t,
        "omega": omega,
        "a0_pump_t": init["a0_pump_t"],
        "a0_probe_t": init["a0_probe_t"],
        "a0_both_t": init["a0_both_t"],
        "A_pump_w": A_pump_w,
        "A_both_w": A_both_w,
        "S_stim": S_stim,
        "A_stim_w": A_stim_w,
    }

    pos = omega > 0.0
    corr["spectral_corr"] = spectral_correlation(S_stim[pos], template[pos])

    if np.isfinite(theory.get("omega_NHR", np.nan)) and np.isfinite(theory.get("omega_DRR", np.nan)):
        wmin = min(theory["omega_NHR"], theory["omega_DRR"]) - 2.0
        wmax = max(theory["omega_NHR"], theory["omega_DRR"]) + 2.0
        spec_mask = (omega > 0.0) & (omega > wmin) & (omega < wmax)
    else:
        lam = cfg.lambda_from_omega(omega)
        spec_mask = (omega > 0.0) & (lam > 0.18) & (lam < 0.50)

    if np.any(spec_mask):
```

```
        domega = float(omega[1] - omega[0])
        spec = spectral_xcorr(S_stim[spec_mask], template[spec_mask], domega)
        corr.update(spec)
        jmax = int(np.argmax(spec["corr_spec_norm"]))
        corr["spec_shift_at_max"] = float(spec["shift_radfs"][jmax])
        corr["spec_maxcorr"] = float(spec["corr_spec_norm"][jmax])

    return result, theory, corr
```

Code 25: main.py: spectral metric and return values

The function finally computes two spectral metrics. The first is the scalar correlation coefficient between the stimulated spectrum and the theory template, namely the quantity defined in Eq. (2.19). The second is the full spectral cross-correlation curve, obtained from the same discrete construction applied on the spectral grid rather than the time grid. From this the code extracts the shift at maximum overlap. If the theory template were centered perfectly on the simulated UV peaks, the maximum would occur close to zero shift. Thus the single-run execution provides not only the propagated fields, but also a compact measure of how well the observed UV structure aligns with the branch prediction. The remaining drawback is that all of these are finite-grid measures, so the precise values of the correlation and shift still depend on the chosen numerical resolution and spectral window.

Probe-wavelength sweep

```
def run_sweep(cfg, lambda_list_um):
    original_lambda_probe = cfg.lambda_probe

    t = make_time_grid(cfg)
    omega = make_omega_grid(t)

    init0 = build_initial_fields(cfg, t)
    res_pump = propagate_uppe(init0["a0_pump_t"], cfg)
    A_pump_w = res_pump["a1_w"]

    out = {
        "lambda_probe_um": [],
        "theory_NHR_um": [],
        "theory_DRR_um": [],
        "sim_NHR_um": [],
        "sim_DRR_um": [],
        "spectral_corr": [],
    }
```

Code 26: main.py: sweep setup and pump reuse

The sweep reuses a single pump-only propagation and then varies only the probe wavelength. This is computationally efficient and also conceptually clean, because the comparison is always made against the same pump background. The only changing external parameter is λ_2, the probe wavelength.

```
    for lam2 in lambda_list_um:
        cfg.lambda_probe = float(lam2)

        init = build_initial_fields(cfg, t)
        res_both = propagate_uppe(init["a0_both_t"], cfg)
        A_both_w = res_both["a1_w"]

        theory = hawking_theory(cfg, lam2)
        S_stim = np.maximum(np.abs(A_both_w) ** 2 - np.abs(A_pump_w) ** 2, 0.0)

        sep = abs(theory["omega_NHR"] - theory["omega_DRR"])
```

```
        w_width = max(0.03, 0.45 * sep)

        wN_sim = extract_peak_near(omega, S_stim, theory["omega_NHR"], width=w_width)
        wB_sim = extract_peak_near(omega, S_stim, theory["omega_DRR"], width=w_width)

        lamN_sim = float(cfg.lambda_from_omega(wN_sim)) if np.isfinite(wN_sim) else np.nan
        lamB_sim = float(cfg.lambda_from_omega(wB_sim)) if np.isfinite(wB_sim) else np.nan

        template = gaussian_template(omega, [theory["omega_NHR"], theory["omega_DRR"]], sigma=0.25)
        pos = omega > 0.0
        c_spec = spectral_correlation(S_stim[pos], template[pos])

        out["lambda_probe_um"].append(float(lam2))
        out["theory_NHR_um"].append(float(theory["lambda_NHR_um"]))
        out["theory_DRR_um"].append(float(theory["lambda_DRR_um"]))
        out["sim_NHR_um"].append(lamN_sim)
        out["sim_DRR_um"].append(lamB_sim)
        out["spectral_corr"].append(c_spec)

    cfg.lambda_probe = original_lambda_probe
    return out
```

Code 27: main.py: branch tracking across the sweep

For each probe wavelength, the code constructs the corresponding theory branches, propagates the pump+probe field, and then extracts the simulated peaks near those theory predictions. The extraction window is not fixed globally; instead it is chosen from the theory branch separation,

$$w_{\mathrm{width}} = \max\bigl(0.03,\ 0.45\,|\omega_{\mathrm{NHR}} - \omega_{\mathrm{DRR}}|\bigr), \tag{2.23}$$

so that the two search windows remain local to their respective theory branches and do not collapse onto one another when the branch separation changes. The function then stores

$$\lambda_2, \quad \lambda^{\mathrm{th}}_{\mathrm{NHR}},\ \lambda^{\mathrm{th}}_{\mathrm{DRR}}, \quad \lambda^{\mathrm{sim}}_{\mathrm{NHR}},\ \lambda^{\mathrm{sim}}_{\mathrm{DRR}}, \tag{2.24}$$

together with the spectral correlation score used in the final comparison panel. Restoring the original probe wavelength at the end keeps the configuration object well-defined after the sweep.

Interpretation of `main.py`

The logic of `main.py` is therefore the following. First, it constructs the two-colour analytic input field. Second, it computes the co-moving theory targets and maps them into the laboratory UV window where the simulation is read out. Third, it compares pump-only and pump+probe propagation in order to isolate the stimulated spectral contribution. Fourth, it evaluates whether the simulated UV bands lie near the predicted NHR and DRR branches for one probe wavelength and then across a full probe-wavelength sweep. In this way the module acts as the bridge between the propagation solver and the final physics figures hence it is where the abstract mathematical branch picture becomes a concrete comparison between predicted and simulated spectral bands.

plots.py

Main visualization: `Hawking_plots`

```python
def Hawking_plots(result_single, theory_single, corr_single, sweep=None, filename="hawking_output.png"):
    t = result_single["t"]
    omega = result_single["omega"]
    c = theory_single["c"]

    lam = _omega_to_lambda_um(omega, c)
    pos = omega > 0.0

    S_pump = np.abs(result_single["A_pump_w"]) ** 2
    S_both = np.abs(result_single["A_both_w"]) ** 2
    S_stim = np.maximum(S_both - S_pump, 0.0)

    idx = np.argsort(lam[pos])
    lam_p = lam[pos][idx]
    Sp = S_pump[pos][idx]
    Sb = S_both[pos][idx]
    Ss = S_stim[pos][idx]

    lam_NHR = theory_single.get("lambda_NHR_um", None)
    lam_DRR = theory_single.get("lambda_DRR_um", None)

    fig, axes = plt.subplots(4, 1, figsize=(10, 12), constrained_layout=True)
```

Code 28: plots.py: extracting the plotted spectra

The plotting routine begins by converting the propagated positive-frequency grid from angular frequency to wavelength,

$$\lambda(\omega) = \frac{2\pi c}{\omega}, \qquad \omega > 0, \tag{2.25}$$

and then restricting attention to the physical positive-frequency sector. The three spectral quantities passed to the figure are

$$S_{\mathrm{pump}}(\omega) = |A_{\mathrm{pump}}(\omega)|^2, \qquad S_{\mathrm{both}}(\omega) = |A_{\mathrm{both}}(\omega)|^2, \qquad S_{\mathrm{stim}}(\omega) = \max\big(S_{\mathrm{both}} - S_{\mathrm{pump}}, 0\big). \tag{2.26}$$

After that, the arrays are sorted by wavelength so that the horizontal axis is visually intuitive. This sorting step is purely for presentation: the simulation itself is performed on the native FFT ω-grid.

```python
    axes[0].plot(t, np.real(result_single["a0_pump_t"]), label="Re[pump] @ z=0")
    axes[0].plot(t, np.real(result_single["a0_probe_t"]), label="Re[probe] @ z=0")
    axes[0].plot(t, np.real(result_single["a0_both_t"]), label="Re[pump+probe] @ z=0", alpha=0.7)
    axes[0].set_xlabel("retarded time τ (fs)")
    axes[0].set_ylabel("Re[a(τ)]")
    axes[0].grid(True, alpha=0.3)
    axes[0].legend(loc="upper right")
```

Code 29: plots.py: panel 1 — input fields in time

Panel 1 shows the real parts of the initial analytic-signal fields at $z = 0$. This panel is not an output metric but an input reference. It makes clear which two carrier-modulated pulses are launched into the solver and how their sum is arranged in the retarded-time window. In other words, the first panel displays the initial condition

$$a(t, 0) = a_{\mathrm{pump}}(t) + a_{\mathrm{probe}}(t) \tag{2.27}$$

whose subsequent propagation generates the UV response shown later.

```
    uv_mask = (lam_p >= 0.18) & (lam_p <= 0.50)
    lam_uv = lam_p[uv_mask]
    Sp_uv = Sp[uv_mask]
    Sb_uv = Sb[uv_mask]
    Ss_uv = Ss[uv_mask]

    axes[1].plot(lam_uv, _normalize(Sp_uv), label="pump-only |A(ω)|^2 (UV norm)")
    axes[1].plot(lam_uv, _normalize(Sb_uv), label="pump+probe |A(ω)|^2 (UV norm)")
    axes[1].plot(lam_uv, _normalize(Ss_uv), label="stimulated (diff) (UV norm)")
    axes[1].set_xlim(0.18, 0.50)
    axes[1].set_xlabel("wavelength λ (um)")
    axes[1].set_ylabel("normalized spectral intensity (UV)")
    axes[1].grid(True, alpha=0.3)
    if lam_NHR is not None:
        axes[1].axvline(lam_NHR, linestyle="--", linewidth=1.2, label="theory NHR")
    if lam_DRR is not None:
        axes[1].axvline(lam_DRR, linestyle=":", linewidth=1.2, label="theory DRR")
    axes[1].legend(loc="upper right")
```

Code 30: plots.py: panel 2 — UV spectrum and theory markers

Panel 2 is the central spectral comparison. The code restricts the wavelength axis to the UV window

$$0.18\,\mu\text{m} \le \lambda \le 0.50\,\mu\text{m}, \tag{2.28}$$

because this is the region where the branch-selected NHR and DRR partners are expected in the present simulation. Each spectrum is normalized separately by the plotting helper, so the panel compares spectral *shape* and peak location rather than absolute power. The vertical lines mark the theory-predicted laboratory wavelengths obtained from the co-moving branch calculation in `main.py`. This panel therefore answers the most direct question in the simulation: whether the stimulated UV signal appears in the same spectral region as the predicted partner branches.

```
    if ("shift_radfs" in corr_single) and ("corr_spec_norm" in corr_single):
        axins = axes[1].inset_axes([0.62, 0.10, 0.35, 0.35])
        axins.plot(corr_single["shift_radfs"], corr_single["corr_spec_norm"])
        if "spec_shift_at_max" in corr_single:
            axins.axvline(corr_single["spec_shift_at_max"], linestyle="--", linewidth=1.0)
        axins.set_title("spectral xcorr", fontsize=8)
        axins.set_xlabel("Δω (rad/fs)", fontsize=7)
        axins.set_ylabel("xcorr", fontsize=7)
        axins.tick_params(axis="both", labelsize=7)
        axins.grid(True, alpha=0.3)
```

Code 31: plots.py: inset in panel 2 — spectral cross-correlation

The inset refines the previous comparison. Instead of showing only the theory markers, it plots the full spectral cross-correlation between the stimulated UV spectrum and the two-peak theory template. If that cross-correlation were maximized at

$$\Delta\omega = 0, \tag{2.29}$$

then the template would already be centered optimally on the observed UV structure. A nonzero maximizing shift means that the simulation and the theory template agree best after a small spectral displacement.

```
    axes[2].plot(corr_single["delay_fs"], corr_single["corr_norm"], label="xcorr(I_NHR, I_DRR)")
    axes[2].set_xlabel("delay (fs)")
    axes[2].set_ylabel("normalized cross-correlation")
    axes[2].grid(True, alpha=0.3)
    axes[2].legend(loc="upper right")
```

Code 32: plots.py: panel 3 — time-domain cross-correlation

Panel 3 visualizes the normalized time-domain cross-correlation defined previously in Eq. (2.20). Since the code normalizes by the maximum absolute value, the panel is a structural metric rather than an absolute-amplitude one. A peak near $\Delta t = 0$ indicates that the two selected UV channels are localized around the same nonlinear interaction event in time.

```
    if sweep is not None:
        lam_probe = np.asarray(sweep["lambda_probe_um"], dtype=float)
        th_N = np.asarray(sweep["theory_NHR_um"], dtype=float)
        th_B = np.asarray(sweep["theory_DRR_um"], dtype=float)
        sim_N = np.asarray(sweep["sim_NHR_um"], dtype=float)
        sim_B = np.asarray(sweep["sim_DRR_um"], dtype=float)

        axes[3].plot(lam_probe, th_N, marker="o", linestyle="--", label="theory NHR")
        axes[3].plot(lam_probe, th_B, marker="o", linestyle=":", label="theory DRR")
        axes[3].plot(lam_probe, sim_N, marker="x", linestyle="-", label="sim NHR")
        axes[3].plot(lam_probe, sim_B, marker="x", linestyle="-", label="sim DRR")
        axes[3].set_xlabel("probe wavelength λ2 (um)")
        axes[3].set_ylabel("UV peak wavelength (um)")
        axes[3].grid(True, alpha=0.3)
        axes[3].legend(loc="upper right")
```

Code 33: plots.py: panel 4 — theory and simulation across the sweep

When a sweep is provided, panel 4 becomes the branch-tracking panel. Its horizontal axis is the input probe wavelength λ_2, and its vertical axis is the UV output wavelength extracted either from theory or from the stimulated spectrum. What is plotted is therefore the comparison

$$\lambda_2 \longmapsto \lambda^{\text{th}}_{\text{NHR}},\ \lambda^{\text{th}}_{\text{DRR}},\ \lambda^{\text{sim}}_{\text{NHR}},\ \lambda^{\text{sim}}_{\text{DRR}}. \tag{2.30}$$

This is the most stringent visual test in the figure, because it checks not only whether the UV response exists at one probe wavelength, but also whether the observed peak locations move with λ_2 in the same way as the theoretical branches.

```
        def _corr(a, b):
            m = np.isfinite(a) & np.isfinite(b)
            if np.sum(m) < 2:
                return np.nan
            A = a[m] - np.mean(a[m])
            B = b[m] - np.mean(b[m])
            den = np.sqrt(np.sum(A * A) * np.sum(B * B))
            if den == 0:
                return np.nan
            return float(np.sum(A * B) / den)

        rN = _corr(sim_N, th_N)
        rB = _corr(sim_B, th_B)
        axes[3].set_title(f"peak-location similarity: corr(NHR)={rN:.3f}, corr(DRR)={rB:.3f}")
```

Code 34: plots.py: panel 4 title — similarity scores

The panel title gives an additional scalar summary. For each branch separately, the code computes a normalized correlation coefficient between the simulated and theoretical wavelength arrays. If the simulated and theoretical curves had exactly the same shape up to an overall affine offset in level, the coefficient would be close to 1. Thus these numbers serve as indicators of how well the simulation tracks the theory across the entire sweep.

```
    else:
        template = theory_single.get("template_uv", None)
        if template is None:
            template = np.zeros_like(Ss)

        Tt = _normalize(template[pos][idx])
        axes[3].plot(lam_p, _normalize(Ss), label="stimulated UV spectrum (norm)")
        axes[3].plot(lam_p, Tt, label="theory template (norm)")
        axes[3].set_xlim(0.18, 0.35)
        axes[3].set_xlabel("wavelength λ (um)")
        axes[3].set_ylabel("normalized amplitude")
        axes[3].grid(True, alpha=0.3)
        axes[3].legend(loc="upper right")

    fig.savefig(filename, dpi=200)
    print(f"Saved {filename}")
    plt.show()
```

Code 35: plots.py: alternative panel 4 when no sweep is given

If no sweep is supplied, the fourth panel is repurposed into a single-run overlay between the stimulated UV spectrum and the normalized theory template. In that case the panel does not compare branch trajectories across probe wavelength, but instead compares the shape of the measured UV response against the two-peak theoretical ansatz at fixed input parameters.

Plotting summary

The logic of `Hawking_plots` is therefore layered. Panel 1 records the launched initial condition. Panel 2 isolates the UV output and places it against the theory-predicted NHR and DRR wavelengths, with an inset quantifying spectral alignment under shifts. Panel 3 tests whether the two selected UV channels are temporally correlated. Panel 4 either tracks theory versus simulation across a probe-wavelength sweep or, if no sweep is provided, overlays the stimulated UV spectrum with the corresponding theory template.

3 Supplementary modules not used in the final results

fme.py

The following two modules are based on Mattan's initial approach to the problem. During the initial study I went over this code as well, so I am including the explanations here for completeness even though these modules are not used in the final stimulated Hawking/backreaction plots.

Spectral building blocks

```python
def k_of_omega(omega, cfg):
    return cfg.n(omega) * np.asarray(omega) / cfg.c

def Kz_FME(omega, k_perp, cfg):
    k = k_of_omega(omega, cfg)
    k = np.asarray(k)
    k_perp = np.asarray(k_perp)
    den = np.where(k == 0.0, np.inf, k)
    return k - (k_perp ** 2) / (2.0 * den)

def Q_FME(omega, k_perp, cfg):
    omega = np.asarray(omega)
    n = np.asarray(cfg.n(omega))
    den = np.where(n == 0.0, np.inf, n)
    return omega / (cfg.c * den)
```

Code 36: fme.py: spectral blocks

This module implements a forward Maxwell-type approximation. The basic spectral quantity is

$$k(\omega) = \beta(\omega) = \frac{n(\omega)\,\omega}{c}, \tag{3.1}$$

which is the same dispersive ingredient used elsewhere in the repository. The function `k_of_omega` simply builds this effective propagation constant from the refractive-index model.

The next quantity,

$$K_z^{\mathrm{FME}}(\omega, k_\perp) = k(\omega) - \frac{k_\perp^2}{2\,k(\omega)}, \tag{3.2}$$

is the forward longitudinal wave number appearing in the paraxial or forward Maxwell approximation. It comes from expanding

$$k_z = \sqrt{k(\omega)^2 - k_\perp^2} \tag{3.3}$$

for small transverse wave number $k_\perp$, namely

$$k_z \approx k - \frac{k_\perp^2}{2k}. \tag{3.4}$$

So this object is the longitudinal propagation operator for a forward-propagating mode with frequency ω and transverse structure $k_\perp$.

The quantity

$$Q^{\mathrm{FME}}(\omega, k_\perp) = \frac{\omega}{c\,n(\omega)} \tag{3.5}$$

is the spectral coupling factor multiplying the polarization source. In this reduced Maxwell form, the

material response is therefore split into two pieces: a propagation term K_z^{FME} and a source prefactor Q^{FME}. Although this module is not part of the final Hawking code, it is useful because it shows that the same dispersive medium can be written in a forward Maxwell language rather than only in the analytic-signal UPPE language.

Forward propagation right-hand side

```python
def rhs_dE_dz(Ehat, omega, k_perp, P_hat, cfg):
    Kz = Kz_FME(omega, k_perp, cfg)
    term_field = 1j * Kz * Ehat
    if P_hat is None:
        return term_field
    Q = Q_FME(omega, k_perp, cfg)
    source = 1j * Q * P_hat / (2.0 * cfg.eps0)
    return term_field + source
```

Code 37: fme.py: forward propagation RHS

The actual forward equation implemented here is

$$\frac{\partial \hat{E}}{\partial z} = i\, K_z^{\mathrm{FME}}(\omega, k_\perp)\, \hat{E} + i\, \frac{Q^{\mathrm{FME}}(\omega, k_\perp)}{2\varepsilon_0}\, \hat{P}, \tag{3.6}$$

where $\hat{E}(\omega)$ is the spectral electric field and $\hat{P}(\omega)$ is the spectral polarization source. Thus the structure is again that of a diagonal linear propagation term plus an optional driven term. At the level of numerical architecture this resembles the UPPE solver, but the object being propagated here is a Maxwell field $\hat{E}$, and the source is an externally supplied polarization $\hat{P}$, not the analytic-signal Kerr nonlinearity used in the main simulation.

fwm.py

Spectral pump intensity model

```python
def Pump_intesity_omegadomain(omega, cfg):
    omega = np.asarray(omega)
    I_w = np.zeros_like(omega)
    for i, w in enumerate(omega):
        if w == 0:
            I_w[i] = 0
        else:
            I_w[i] = cfg.I_pump * cfg.tau0 * (np.pi * cfg.tau0 * omega[i]) / (np.sinh(np.pi * cfg.tau0 * omega[i] * 0.5))
    return I_w
```

Code 38: fwm.py: pump intensity in frequency space

This function provides a closed-form spectral envelope for the pump in a reduced mixing model. The structure

$$I_\omega \propto \tau_0 \frac{\pi \tau_0 \omega}{\sinh\left(\frac{\pi \tau_0 \omega}{2}\right)} \tag{3.7}$$

is the familiar Fourier-domain shape associated with a localized hyperbolic-secant-type profile. Thus, rather than propagating the full pump dynamically as in the main UPPE simulation, this reduced model inserts a fixed spectral pump weight directly in frequency space.

Co-moving frequency in the reduced model

```python
def omega_prime(omega, cfg):
    if cfg.truncate_beta:
        return cfg.sign * cfg.L_source * omega ** 2
    return omega * (1 + cfg.n(omega) / cfg.n_g)
```

Code 39: fwm.py: omega prime mapping

This is the reduced model's version of a co-moving invariant. In the truncated case the code replaces the dispersion by an effective quadratic law,

$$\omega' \sim \operatorname{sign} L_{\text{source}}\, \omega^2, \tag{3.8}$$

while in the non-truncated version it uses

$$\omega' = \omega \left(1 + \frac{n(\omega)}{n_g}\right). \tag{3.9}$$

The precise form differs from the UPPE treatment, but conceptually the role is the same: ω' labels the branches that can mix in the moving-frame scattering picture.

Runge–Kutta integrator

```python
def RK4(omega_prime, A0_w, u_t, zeta, h, cfg):
    omega_prime = np.asarray(omega_prime)
    def RHS(A_w, zeta):
        A_t = np.fft.ifft(A_w)
        NL_t = u_t * A_t
        NL_w = np.fft.fft(NL_t)
        return 1j * omega_prime * A_w + 1j * cfg.NL_source * NL_w
    k1 = RHS(A0_w, zeta)
    k2 = RHS(A0_w + 0.5*h*k1, zeta + 0.5*h)
    k3 = RHS(A0_w + 0.5*h*k2, zeta + 0.5*h)
    k4 = RHS(A0_w + h*k3, zeta + h)
    A1_w = A0_w + (h/6.0)*(k1 + 2*k2 + 2*k3 + k4)
    return A1_w
```

Code 40: fwm.py: RK4 step

This reduced model is evolved as an ordinary differential equation in the variable ζ. The right-hand side has the form

$$\frac{dA_\omega}{d\zeta} = i\,\omega'(\omega)\,A_\omega + i\,N_{\mathrm{L}}\,\mathcal{F}\left[u_t\,\mathcal{F}^{-1}(A_\omega)\right], \tag{3.10}$$

where u_t is the time-domain potential or pump profile and N_{L} is the reduced nonlinear-source strength. The update is then performed with the classical fourth-order Runge–Kutta scheme,

$$A_1 = A_0 + \frac{h}{6}\big(k_1 + 2k_2 + 2k_3 + k_4\big). \tag{3.11}$$

This module is therefore structurally an ODE solver in ζ, unlike the split-step spectral propagator used in the main UPPE code.

Four-wave mixing link to the main simulation

In the main UPPE simulation, the Kerr cubic term generates new frequency components through mixing of pump and probe bands. In frequency space, this is four-wave mixing implemented numerically as FFT-domain convolution of products in time. The partner UV bands identified as NHR and DRR correspond to specific mixing channels organized by the co-moving invariant ω'. The simulation makes this visible by using a seeded probe and plotting the stimulated difference spectrum; the sweep then checks whether the extracted UV peaks track the predicted ω'-mapped branches as λ_2 changes.

Analytic bound-state style functions

```python
def energy(cfg, n: int):
    return -(cfg.L * cfg.I_ratio/(2*cfg.L_d)) * (cfg.s - n)**2

def psi(cfg, n, x):
    return (1 / np.cosh(np.sqrt(cfg.I_ratio)*x)**(cfg.s - n) *
           eval_jacobi(n, cfg.s - n, cfg.s - n, np.tanh(np.sqrt(cfg.I_ratio)*x)))
```

Code 41: fwm.py: energy and eigenfunction

These functions define an analytic discrete spectrum and its associated eigenfunctions in the reduced model:

$$E_n = -\frac{L\,I_{\mathrm{ratio}}}{2L_d}\,(s-n)^2, \tag{3.12}$$

and

$$\psi_n(x) = \frac{1}{\cosh\left(\sqrt{I_{\text{ratio}}}\, x\right)^{s-n}} P_n^{(s-n,s-n)}\left(\tanh\left(\sqrt{I_{\text{ratio}}}\, x\right)\right). \tag{3.13}$$

These do not enter the main stimulated Hawking/backreaction execution loop; they are included as optional theoretical structure that can be compared qualitatively to mode families if desired.

4 Results

In this report we outlined the idea of simulating the experimental setup carried out by Prof. Leonhardt's group. The aim was not just to produce ultraviolet light in a generic nonlinear process, but to see whether the stimulated UV response follows the branch structure predicted by the co-moving Hamiltonian picture described in the theoretical section.

The plots we have obtained are the following.

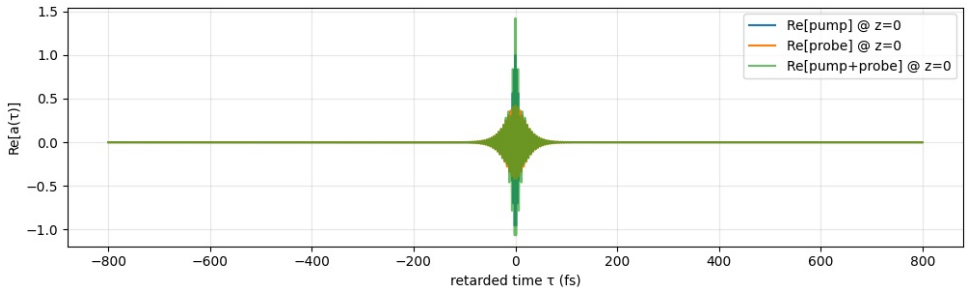

Figure 4: Reduced-model output from Mattan Gelvan's original code.

Figure 4 corresponds directly to Mattan Gelvan's original code. It already shows the general intuition of the problem, namely that a moving optical background can act as a scattering medium for the probe and generate new spectral structure. However, this code is based on a reduced description testing the initial conditions and does not yet contain the final analytic-signal UPPE treatment used in the main part of this report. So this plot should be understood as the original motivation and not as the final quantitative result.

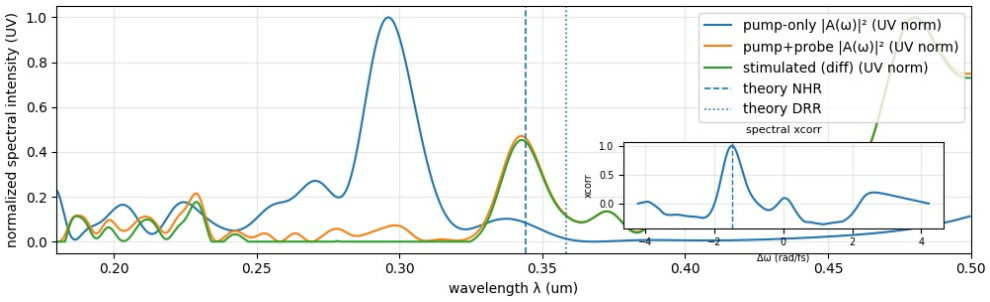

Figure 5: Stimulated ultraviolet spectrum with theory markers for the NHR and DRR branches.

Figure 5 is the most important result in the report. The stimulated ultraviolet spectrum defined in Eq. (2.16) develops spectral weight near the theory-predicted NHR and DRR branches. This is the main evidence that the simulated branch-conversion picture is working in the intended way. The figure therefore supports the interpretation that the ultraviolet output is not a generic by-product of nonlinear broadening, but a structured response tied to the predicted branch locations.

Figure 6 isolates the temporal metric introduced in Eq. (2.20). The near-zero-delay peak indicates that the NHR and DRR channels are generated in the same temporal interaction region rather than being unrelated spectral features. This gives additional support to the interpretation that the two ultraviolet outputs belong to the same branch-conversion process.

Figure 7 is in some sense the strongest test, because it asks whether the extracted ultraviolet peaks move systematically with the probe wavelength in the same way as the theoretical branches. In the figure the theory and simulation both remain in the same ultraviolet region, and the simulated NHR and DRR peaks

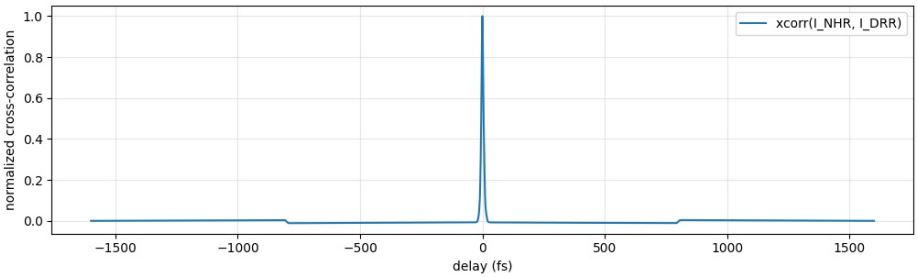

Figure 6: Normalized time-domain cross-correlation between the extracted NHR and DRR ultraviolet channels.

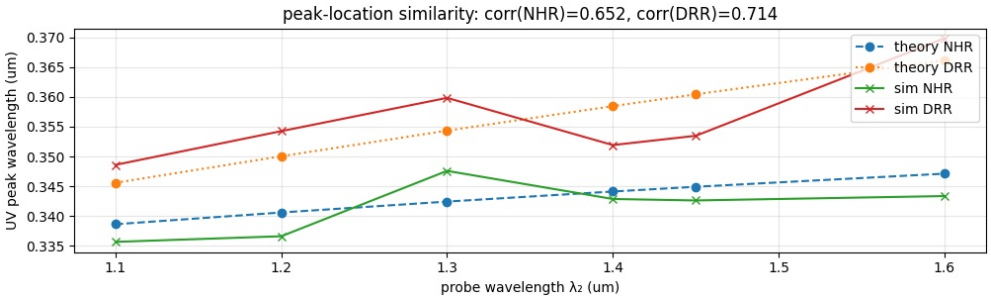

Figure 7: Theory–simulation comparison of NHR and DRR ultraviolet peak wavelengths versus probe wavelength.

track the same overall tendency as the predicted branch curves. So this confirms that the observed UV structure is not a fixed accidental feature of the spectrum, but is tied to the co-moving branch matching of the theory.

Taken together, Figures 5–7 support the following claim. In the present fibre-optical model, a weak seeded probe stimulates ultraviolet output bands which appear near the theory-predicted NHR and DRR branches, remain temporally correlated, and move with the probe wavelength in a way consistent with the branch structure of the co-moving Hamiltonian picture [7].

However, one shortcoming of this simulation is that we were not able to carry out the full paper-like numerical run under the finest grid and propagation conditions. In particular, the final results shown here were obtained on the numerically stable fast grid rather than on the heavier paper-like grid. Concretely, instead of using

$$N_t = 2^{15}, \qquad T = 3176 \text{ fs}, \qquad dz = 0.5\ \mu\text{m}, \tag{4.1}$$

we used

$$N_t = 2^{13}, \qquad T = 1600 \text{ fs}, \qquad dz = 10\ \mu\text{m}. \tag{4.2}$$

Therefore the actual discretization used in the reported simulation was

$$\Delta t = \frac{T}{N_t} \approx 0.195 \text{ fs}, \qquad \Delta\omega = \frac{2\pi}{T} \approx 3.93 \times 10^{-3} \text{ rad/fs}, \qquad \omega_{\text{Ny}} = \frac{\pi}{\Delta t} \approx 16.1 \text{ rad/fs}, \tag{4.3}$$

whereas the paper-like grid would have given

$$\Delta t \approx 0.0969 \text{ fs}, \qquad \Delta\omega \approx 1.98 \times 10^{-3} \text{ rad/fs}, \qquad \omega_{\text{Ny}} \approx 32.4 \text{ rad/fs}. \tag{4.4}$$

Similarly, with total propagation length fixed at

$$z_{\text{total}} = 7000\ \mu\text{m}, \tag{4.5}$$

the fast run uses only

$$N_z = \frac{z_{\text{total}}}{dz} = 700 \tag{4.6}$$

propagation steps, while the paper-like run would require

$$N_z = 14000. \tag{4.7}$$

Hence the present simulation has coarser temporal resolution, coarser spectral resolution, a lower ultraviolet cutoff, and a less refined split-step propagation in z. Mathematically, this means that the present results should be interpreted as a stable qualitative demonstration of branch conversion and stimulated UV generation, but not yet as a numerically converged reproduction of the full paper-level simulation.

In addition, we also did not use a fully calibrated experimental dispersion relation, exact SI-calibrated nonlinear material parameters, the full spatial structure of the real fibre, or delayed nonlinear effects beyond the instantaneous Kerr-type model used here. Instead, we used a one-dimensional analytic-signal propagation model with an effective refractive-index function $n(\omega)$, scaled nonlinear strength, and sech input pulses. This should still capture the main mechanism of branch conversion and stimulated ultraviolet generation, but for a complete quantitative comparison with experiment one would need to repeat the analysis on the finer paper-like grid together with the exact measured fibre parameters and full experimental operating conditions.

Theoretically, we also outlined a more general conjecture. Starting from the spirit of Zel'dovich's superradiant ideas, the suggestion is that Hawking-like spectral leakage may not be restricted only to gravitational systems, but may arise more generally in classical systems whose governing partial differential equations admit a moving inhomogeneous background, a conserved co-moving invariant, and a turning-point branch-conversion structure. In that sense the present optical system should be viewed not as an imitation in a loose metaphorical sense, but as a concrete mathematical laboratory for studying the horizon mechanism itself.

References

[1] L. M. Procopio, R. Aguero-Santacruz, D. Bermudez, and U. Leonhardt, *Observation of the backreaction of stimulated Hawking radiation in an optical analogue* (to be published).

[2] T. G. Philbin, C. Kuklewicz, S. Robertson, S. Hill, F. König, and U. Leonhardt, "Fiber-Optical Analog of the Event Horizon," *Science* **319**, 1367–1370 (2008).

[3] Ya. B. Zel'dovich, "Generation of Waves by a Rotating Body," *JETP Letters* **14**, 180–181 (1971).

[4] Ya. B. Zel'dovich, "Amplification of Cylindrical Electromagnetic Waves Reflected from a Rotating Body," *Soviet Physics JETP* **35**, 1085–1087 (1972).

[5] S. W. Hawking, "Particle Creation by Black Holes," *Communications in Mathematical Physics* **43**, 199–220 (1975).

[6] D. Bermudez and U. Leonhardt, "Hawking spectrum for a fiber-optical analog of the event horizon," *Physical Review A* **93**, 053820 (2016).

[7] J. Drori, Y. Rosenberg, D. Bermudez, Y. Silberberg, and U. Leonhardt, "Observation of Stimulated Hawking Radiation in an Optical Analogue," *Physical Review Letters* **122**, 010404 (2019).

[8] S. Robertson and U. Leonhardt, "Frequency shifting at fiber-optical event horizons: The effect of Raman deceleration," *Physical Review A* **81**, 063835 (2010).

[9] C. Adami, "Paradox No More: How Stimulated Emission of Radiation Preserves Information Absorbed by Black Holes," *arXiv preprint* arXiv:2502.05642 [gr-qc] (2025).