frontiers | Frontiers in Neuroinformatics

TECHNOLOGY AND CODE
published: 27 May 2022
doi: 10.3389/fninf.2022.835657

NNMT: Mean-Field Based Analysis Tools for Neuronal Network Models

Moritz Layer[1,2]*, *Johanna Senk*[1], *Simon Essink*[1,2], *Alexander van Meegen*[1,3], *Hannah Bos*[1] *and Moritz Helias*[1,4]

[1] *Institute of Neuroscience and Medicine (INM-6) and Institute for Advanced Simulation (IAS-6) and JARA-Institute Brain Structure-Function Relationships (INM-10), Jülich Research Centre, Jülich, Germany,* [2] *RWTH Aachen University, Aachen, Germany,* [3] *Institute of Zoology, Faculty of Mathematics and Natural Sciences, University of Cologne, Cologne, Germany,* [4] *Department of Physics, Faculty 1, RWTH Aachen University, Aachen, Germany*

Mean-field theory of neuronal networks has led to numerous advances in our analytical and intuitive understanding of their dynamics during the past decades. In order to make mean-field based analysis tools more accessible, we implemented an extensible, easy-to-use open-source Python toolbox that collects a variety of mean-field methods for the leaky integrate-and-fire neuron model. The Neuronal Network Mean-field Toolbox (NNMT) in its current state allows for estimating properties of large neuronal networks, such as firing rates, power spectra, and dynamical stability in mean-field and linear response approximation, without running simulations. In this article, we describe how the toolbox is implemented, show how it is used to reproduce results of previous studies, and discuss different use-cases, such as parameter space explorations, or mapping different network models. Although the initial version of the toolbox focuses on methods for leaky integrate-and-fire neurons, its structure is designed to be open and extensible. It aims to provide a platform for collecting analytical methods for neuronal network model analysis, such that the neuroscientific community can take maximal advantage of them.

Keywords: mean-field theory, (spiking) neuronal network, integrate-and-fire neuron, open-source software, parameter space exploration, (hybrid) modeling, python, computational neuroscience

OPEN ACCESS

Edited by:
John David Griffiths, University of Toronto, Canada

Reviewed by:
Caglar Cakan, Technical University of Berlin, Germany
Richard Gast, Max Planck Institute for Human Cognitive and Brain Sciences, Germany

***Correspondence:**
Moritz Layer
m.layer@fz-juelich.de

Received: *14 December 2021*
Accepted: *17 March 2022*
Published: *27 May 2022*

Citation:
Layer M, Senk J, Essink S, van Meegen A, Bos H and Helias M (2022) NNMT: Mean-Field Based Analysis Tools for Neuronal Network Models. Front. Neuroinform. 16:835657. doi: 10.3389/fninf.2022.835657

1. INTRODUCTION

Biological neuronal networks are composed of large numbers of recurrently connected neurons, with a single cortical neuron typically receiving synaptic inputs from thousands of other neurons (Braitenberg and Schüz, 1998; DeFelipe et al., 2002). Although the inputs of distinct neurons are integrated in a complex fashion, such large numbers of weak synaptic inputs imply that average properties of entire populations of neurons do not depend strongly on the contributions of individual neurons (Amit and Tsodyks, 1991). Based on this observation, it is possible to develop analytically tractable theories of population properties, in which the effects of individual neurons are averaged out and the complex, recurrent input to individual neurons is replaced by a self-consistent effective input (reviewed, e.g., in Gerstner et al., 2014). In classical physics terms (e.g., Goldenfeld, 1992), this effective input is called *mean-field*, because it is the self-consistent mean of a *field*, which here is just another name for the input the neuron is receiving. The term *self-consistent* refers to the fact that the population of neurons that receives the effective input is the same that contributes to this very input in a recurrent fashion: the population's output determines its input and vice-versa. The stationary statistics of the effective input therefore can be found in a

self-consistent manner: the input to a neuron must be set exactly such that the caused output leads to the respective input.

Mean-field theories have been developed for many different kinds of synapse, neuron, and network models. They have been successfully applied to study average population firing rates (van Vreeswijk and Sompolinsky, 1996, 1998; Amit and Brunel, 1997b), and the various activity states a network of spiking neurons can exhibit, depending on the network parameters (Amit and Brunel, 1997a; Brunel, 2000; Ostojic, 2014), as well as the effects that different kinds of synapses have on firing rates (Fourcaud and Brunel, 2002; Lindner, 2004; Schuecker et al., 2015; Schwalger et al., 2015; Mattia et al., 2019). They have been used to investigate how neuronal networks respond to external inputs (Lindner and Schimansky-Geier, 2001; Lindner and Longtin, 2005), and they explain why neuronal networks can track external input on much faster time scales than a single neuron could (van Vreeswijk and Sompolinsky, 1996, 1998). Mean-field theories allow studying correlations of neuronal activity (Sejnowski, 1976; Ginzburg and Sompolinsky, 1994; Lindner et al., 2005; Trousdale et al., 2012) and were able to reveal why pairs of neurons in random networks, despite receiving a high proportion of common input, can show low output correlations (Hertz, 2010; Renart et al., 2010; Tetzlaff et al., 2012; Helias et al., 2014), which for example has important implication for information processing. They describe pair-wise correlations in network with spatial organization (Rosenbaum and Doiron, 2014; Rosenbaum et al., 2017; Dahmen et al., 2022) and can be generalized to correlations of higher orders (Buice and Chow, 2013). Mean-field theories were utilized to show that neuronal networks can exhibit chaotic dynamics (Sompolinsky et al., 1988; van Vreeswijk and Sompolinsky, 1996, 1998), in which two slightly different initial states can lead to totally different network responses, which has been linked to the network's memory capacity (Toyoizumi and Abbott, 2011; Schuecker et al., 2018). Most of the results mentioned above have been derived for networks of either rate, binary, or spiking neurons of a linear integrate-and-fire type. But various other models have been investigated with similar tools as well; for example, just to mention a few, Hawkes processes, non-linear integrate-and-fire neurons (Brunel and Latham, 2003; Fourcaud-Trocmé et al., 2003; Richardson, 2007, 2008; Grabska-Barwinska and Latham, 2014; Montbrió et al., 2015), or Kuramoto-type models (Stiller and Radons, 1998; van Meegen and Lindner, 2018). Additionally, there is an ongoing effort showing that many of the results derived for distinct models are indeed equivalent and that those models can be mapped to each other under certain circumstances (Ostojic and Brunel, 2011; Grytskyy et al., 2013; Senk et al., 2020).

Other theories for describing mean population rates in networks with spatially organized connectivity, based on taking a continuum limit, have been developed. These theories, known as neural field theories, have deepened our understanding of spatially and temporally structured activity patterns emerging in cortical networks, starting with the seminal work by Wilson and Cowan (1972, 1973), who investigated global activity patterns, and Amari (1975, 1977), who studied stable localized neuronal activity. They were successfully applied to explain hallucination patterns (Ermentrout and Cowan, 1979; Bressloff et al., 2001), as well as EEG and MEG rhythms (Nunez, 1974; Jirsa and Haken, 1996, 1997). The neural field approach has been used to model working memory (Laing et al., 2002; Laing and Troy, 2003), motion perception (Giese, 2012), cognition (Schöner, 2008), and more; for extensive reviews of the literature, we refer the reader to Coombes (2005), Bressloff (2012), and Coombes et al. (2014).

Clearly, analytical theories have contributed to our understanding of neuronal networks and they provide a plethora of powerful and efficient methods for network model analysis. Comparing the predictions of analytical theories to simulations, experimental data, or other theories necessitates a numerical implementation applicable to various network models, depending on the research question. Such an implementation is often far from straightforward and at times requires investing substantial time and effort. Commonly, such tools are implemented as the need arises, and their reuse is not organized systematically and restricted to within a single lab. This way, not only are effort and costs spent by the neuroscientific community duplicated over and over again, but also are many scientists deterred from taking maximal advantage of those methods although they might open new avenues for investigating their research questions.

In order to make analytical tools for neuronal network model analysis accessible to a wider part of the neuroscientific community, and to create a platform for collecting well-tested and validated implementations of such tools, we have developed the Python toolbox NNMT (Layer et al., 2021), short for Neuronal Network Mean-field Toolbox. We would like to emphasize that NNMT is not a simulation tool; NNMT is a collection of numerically solved mean-field equations that directly relate the parameters of a microscopic network model to the statistics of its dynamics. NNMT has been designed to fit the diversity of mean-field theories, and the key features we are aiming for are modularity, extensibility, and a simple usability. Furthermore, it features an extensive test suite to ensure the validity of the implementations as well as a comprehensive user documentation. The current version of NNMT mainly comprises tools for investigating networks of leaky integrate-and-fire neurons as well as some methods for studying binary neurons and neural field models. The toolbox is open-source and publicly available on GitHub.[1]

In the following, we present the design considerations that led to the structure and implementation of NNMT as well as a representative set of use cases. Section 2 first introduces its architecture. Section 3 then explains its usage by reproducing previously published network model analyses from Schuecker et al. (2015), Bos et al. (2016), Sanzeni et al. (2020), and Senk et al. (2020). Section 4 compares NNMT to other available toolboxes for neuronal network model analysis, discusses its use cases from a more general perspective, indicates current limitations and prospective advancements of NNMT, and explains how new tools can be contributed.

[1] https://github.com/INM-6/nnmt

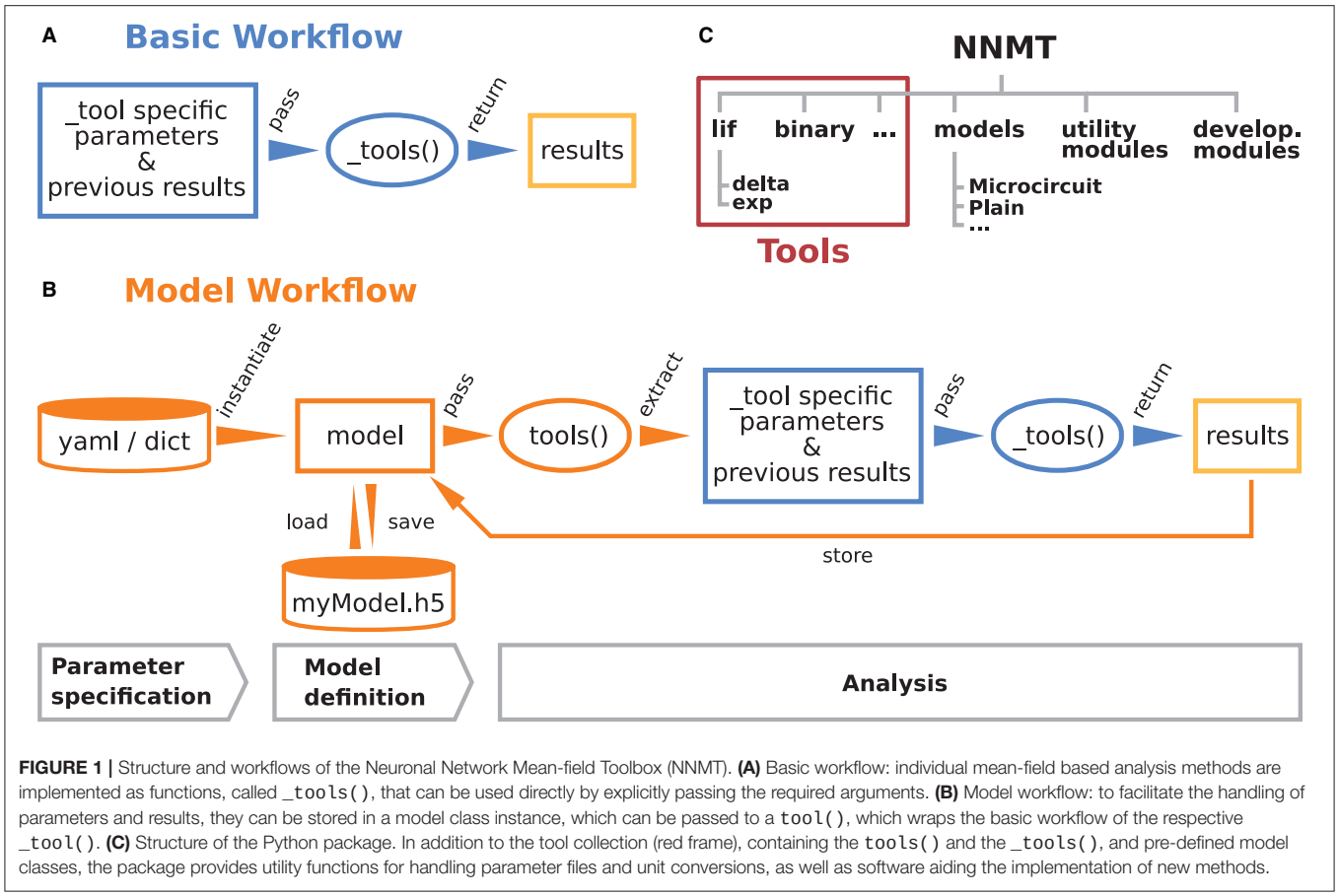

FIGURE 1 | Structure and workflows of the Neuronal Network Mean-field Toolbox (NNMT). **(A)** Basic workflow: individual mean-field based analysis methods are implemented as functions, called `_tools()`, that can be used directly by explicitly passing the required arguments. **(B)** Model workflow: to facilitate the handling of parameters and results, they can be stored in a model class instance, which can be passed to a `tool()`, which wraps the basic workflow of the respective `_tool()`. **(C)** Structure of the Python package. In addition to the tool collection (red frame), containing the `tools()` and the `_tools()`, and pre-defined model classes, the package provides utility functions for handling parameter files and unit conversions, as well as software aiding the implementation of new methods.

```
# basic workflow
result = nnmt.<submodule>.<_tool>(*args, **kwargs)

# model workflow
my_model = nnmt.models.<model>(
    <network_params>, <analysis_params>)
result = nnmt.<submodule>.<tool>(my_model)
```

Listing 1: The two modes of using NNMT: In the basic workflow (top), quantities are calculated by passing all required arguments directly to the underscored tool functions available in the submodules of NNMT. In the model workflow (bottom), a model class is instantiated with parameter sets and the model instance is passed to the non-underscored tool functions which automatically extract the relevant parameters.

2. WORKFLOWS AND ARCHITECTURE

What are the requirements a package for collecting analytical methods for neuronal network model analysis needs to fulfill? To begin with, it should be adaptable and modular enough to accommodate many and diverse analytical methods while avoiding code repetition and a complex interdependency of package components. It should enable the application of the collected algorithms to various network models in a simple and transparent manner. It should make the tools easy to use for new users, while also providing experts with direct access to all parameters and options. Finally, the methods need to be thoroughly tested and well documented.

These are the main considerations that guided the development of NNMT. **Figures 1A,B** illustrate how the toolbox can be used in to two different workflows, depending on the preferences and goals of the user. In the *basic workflow* the individual method implementations called *tools* are directly accessed, whereas the *model workflow* provides additional functionality for the handling of parameters and results.

2.1. Basic Workflow

The core of NNMT is a collection of low-level functions that take specific parameters (or pre-computed results) as input arguments and return analytical results of network properties. In **Figure 1A**, we refer to such basic functions as `_tools()`, as their names always start with an underscore. We term this lightweight approach of directly using these functions the basic workflow. The top part of **Listing 1** demonstrates this usage; for example, the quantity to be computed could be the mean firing rate of a neuronal population and the arguments could be parameters which define neuron model and external drive. While the basic workflow gives full flexibility and direct access to every

parameter of the calculation, it remains the user's responsibility to insert the arguments correctly, e.g., in the right units.

2.2. Model Workflow

The model workflow is a convenient wrapper of the basic workflow (**Figure 1B**). A *model* in this context is an object that stores a larger set of parameters and can be passed directly to a `tool()`, the non-underscored wrapper of the respective `_tool()`. The `tool()` automatically extracts the relevant parameters from the model, passes them as arguments to the corresponding core function `_tool()`, returns the results, and stores them in the model. The bottom part of **Listing 1** shows how a model is initialized with parameters and then passed to a `tool()` function.

Models are implemented as Python classes and can be found in the submodule `nnmt.models`. We provide the class `nnmt.models.Network` as a parent class and a few child classes which inherit the generic methods and properties but are tailored to specific network models; custom models can be created straightforwardly. The parameters distinguish network parameters, which define neuron models and network connectivity, and analysis parameters; an example for an analysis parameter is a frequency range over which a function is evaluated. Upon model instantiation, parameter sets defining values and corresponding units are passed as Python dictionaries or `yaml` files. The model constructor takes care of reading in these parameters, computing dependent parameters from the imported parameters, and converting all units to SI units for internal computations. Consequently, the parameters passed as arguments and the functions for computing dependent parameters of a specific child class need to be aligned. This design encourages a clear separation between a concise set of base parameters and functionality that transforms these parameters to the generic (vectorized) format that the tools work with. To illustrate this, consider the weight matrix of a network of excitatory and inhibitory neuron populations in which all excitatory connections have the same weight and all inhibitory ones another weight. As argument one could pass just a tuple of two different weight values and the corresponding model class would take care of constructing the full weight matrix. This happens in the example presented in Section 3.2.2: The parameter file `network_params_microcircuit.yaml` contains the excitatory synaptic weight and the ratio of inhibitory to excitatory weights. On instantiation, the full weight matrix is constructed from these two parameters, following the rules defined in `nnmt.models.Microcircuit`.

When a `tool()` is called, it checks whether the provided model object contains all required parameters and previously computed results. Then the `tool()` extracts the required arguments, calls the respective `_tool()`, and caches and returns the result. If the user attempts to compute the same property twice, using identical parameters, the `tool()` will retrieve the already computed result from the model's cache and return that value. Results can be exported to an HDF5 file and also loaded.

Using the model workflow instead of the basic workflow comes with the initial overhead of choosing a suitable combination of parameters and a model class, but has the advantages of a higher level of automation with built-in mechanisms for checking correctness of input (e.g., regarding units), reduced redundancy, and the options to store and load results. Both modes of using the toolbox can also be combined.

2.3. Structure of the Toolbox

The structure of the Python package NNMT is depicted in **Figure 1C**. It is subdivided into submodules containing the tools (e.g., `nnmt.lif.exp`, or `nnmt.binary`), the model classes (`nnmt.models`), helper routines for handling parameter files and unit conversions, as well as modules that collect reusable code employed in implementations for multiple neuron models (cf. Section 4.4). The tools are organized in a modular, extensible fashion with a streamlined hierarchy. To give an example, a large part of the currently implemented tools apply to networks of leaky integrate-and-fire (LIF) neurons, and they are located in the submodule `nnmt.lif`. The mean-field theory for networks of LIF neurons distinguishes between neurons with instantaneous synapses, also called delta synapses, and those with exponentially decaying post-synaptic currents. Similarly, the submodule for LIF neurons is split further into the two submodules `nnmt.lif.delta` and `nnmt.lif.exp`. NNMT also collects different implementations for computing the same quantity using different approximations or numerics, allowing for a comparison of different approaches.

Apart from the core package, NNMT comes with an extensive online documentation,[2] including a quickstart tutorial, all examples presented in this paper, a complete documentation of all tools, as well as a guide for contributors.

Furthermore, we provide an extensive test suite that validates the tools by checking them against previously published results and alternative implementations where possible. This ensures that future improvements of the numerics do not break the tools.

3. HOW TO USE THE TOOLBOX

In this section, we demonstrate the practical use of NNMT by replicating a variety of previously published results. The examples presented have been chosen to cover a broad range of common use cases and network models. We include analyses of both stationary and dynamic network features, as mean-field theory is typically divided into two parts: stationary theory, which describes time-independent network properties of systems in a stationary state, and dynamical theory, which describes time-dependent network properties. Additionally, we show how to use the toolbox to map a spiking to a simpler rate model, as well as how to perform a linear stability analysis. All examples, including the used parameter files, are part of the online documentation.[2]

3.1. Installation and Setup

The toolbox can be either installed using pip:

```
pip install nnmt
```

or by installing it directly from the repository, which is described in detail in the online

[2]https://nnmt.readthedocs.io/

documentation. After the installation, the module can be imported:

```
import nnmt
```

3.2. Stationary Quantities

3.2.1. Response Nonlinearities

Networks of excitatory and inhibitory neurons (EI networks, **Figure 2A**) are widely used in computational neuroscience (Gerstner et al., 2014), e.g., to show analytically that a balanced state featuring asynchronous, irregular activity emerges dynamically in a broad region of the parameter space (van Vreeswijk and Sompolinsky, 1996, 1998; Brunel, 2000; Hertz, 2010; Renart et al., 2010). Remarkably, such balance states emerge in inhibition dominated networks for a variety of neuron models if the indegree is large, $K \gg 1$, and the weights scale as $J \propto 1/\sqrt{K}$ (Sanzeni et al., 2020; Ahmadian and Miller, 2021). Furthermore, in a balanced state, a network responds linearly to external input in the limit $K \to \infty$ (van Vreeswijk and Sompolinsky, 1996, 1998; Brunel, 2000; Sanzeni et al., 2020; Ahmadian and Miller, 2021). How do EI networks of LIF neurons respond to external input at finite indegrees? Sanzeni et al. (2020) uncover five different types of nonlinearities in the network response depending on the network parameters. Here, we show how to use the toolbox to reproduce their result (**Figures 2B–F**).

The network consists of two populations, E and I, of identical LIF neurons with instantaneous (*delta*) synapses (Gerstner et al., 2014). The subthreshold dynamics of the membrane potential V_i of neuron i obeys

$$\tau_\mathrm{m} \dot{V}_i = -V_i + RI_i \,, \tag{1}$$

where τ_m denotes the membrane time constant, R the membrane resistance, and I_i the input current. If the membrane potential exceeds a threshold V_th, a spike is emitted and the membrane voltage is reset to the reset potential V_0 and clamped to this value during the refractory time τ_r. After the refractory period, the dynamics continue according to Equation (1). For instantaneous synapses, the input current is given by

$$RI_i(t) = \tau_\mathrm{m} \sum_j J_{ij} \sum_k \delta(t - t_{j,k} - d_{ij}) \,, \tag{2}$$

where J_{ij} is the synaptic weight from presynaptic neuron j to postsynaptic neuron i (with $J_{ij} = 0$ if there is no synapse), the $t_{j,k}$ are the spike times of neuron j, and d_{ij} is a synaptic delay (in this example $d_{ij} = d$ for all pairs of neurons). In total, there are N_E and N_I neurons in the respective populations. Each neuron is connected to a fixed number of randomly chosen presynaptic neurons (fixed in-degree); additionally, all neurons receive external input from independent Poisson processes with rate ν_X. The synaptic weights and in-degrees of recurrent and external connections are population-specific:

$$\boldsymbol{J} = \begin{pmatrix} J_\mathrm{EE} & -J_\mathrm{EI} \\ J_\mathrm{IE} & -J_\mathrm{II} \end{pmatrix}, \; \boldsymbol{J}_\mathrm{ext} = \begin{pmatrix} J_\mathrm{EX} \\ J_\mathrm{IX} \end{pmatrix}, \\ \boldsymbol{K} = \begin{pmatrix} K_\mathrm{EE} & K_\mathrm{EI} \\ K_\mathrm{IE} & K_\mathrm{II} \end{pmatrix}, \; \boldsymbol{K}_\mathrm{ext} = \begin{pmatrix} K_\mathrm{EX} \\ K_\mathrm{IX} \end{pmatrix}. \tag{3}$$

All weights are positive, implying an excitatory external input.

The core idea of mean-field theory is to approximate the input to a neuron as Gaussian white noise $\xi(t)$ with mean $\langle \xi(t) \rangle = \mu$ and noise intensity $\langle \xi(t)\xi(t') \rangle = \tau_\mathrm{m}\sigma^2 \delta(t - t')$. This approximation is well-suited for asynchronous, irregular network states (van Vreeswijk and Sompolinsky, 1996, 1998; Amit and Brunel, 1997b). For a LIF neuron driven by such Gaussian white noise, the firing rate is given by (Siegert, 1951; Tuckwell, 1988; Amit and Brunel, 1997b)

$$\phi(\mu, \sigma) = \left(\tau_\mathrm{r} + \tau_\mathrm{m} \sqrt{\pi} \int_{\widetilde{V}_0(\mu,\sigma)}^{\widetilde{V}_\mathrm{th}(\mu,\sigma)} e^{s^2} (1 + \mathrm{erf}(s)) \mathrm{d}s \right)^{-1}, \tag{4}$$

where the rescaled reset- and threshold-voltages are

$$\widetilde{V}_0(\mu, \sigma) = \frac{V_0 - \mu}{\sigma}, \qquad \widetilde{V}_\mathrm{th}(\mu, \sigma) = \frac{V_\mathrm{th} - \mu}{\sigma}. \tag{5}$$

The first term in Equation (4) is the refractory period and the second term is the mean first-passage time of the membrane voltage from reset to threshold. The mean and the noise intensity of the input to a neuron in a population $a \in \{E, I\}$, which control the mean first-passage time through Equation (5), are determined by (Amit and Brunel, 1997b)

$$\mu_a = \tau_\mathrm{m} (J_{a\mathrm{E}} K_{a\mathrm{E}} \nu_\mathrm{E} - J_{a\mathrm{I}} K_{a\mathrm{I}} \nu_\mathrm{I} + J_{a\mathrm{X}} K_{a\mathrm{X}} \nu_\mathrm{X}) \,, \tag{6}$$

$$\sigma_a^2 = \tau_\mathrm{m} (J_{a\mathrm{E}}^2 K_{a\mathrm{E}} \nu_\mathrm{E} + J_{a\mathrm{I}}^2 K_{a\mathrm{I}} \nu_\mathrm{I} + J_{a\mathrm{X}}^2 K_{a\mathrm{X}} \nu_\mathrm{X}) \,, \tag{7}$$

respectively, where each term reflects the contribution of one population, with the corresponding firing rates of the excitatory ν_E, inhibitory ν_I, and external population ν_X. Note that we use the letters $i, j, k, \ldots$ to index single neurons and $a, b, c, \ldots$ to index neuronal populations. Both μ_a and σ_a depend on the firing rate of the neurons ν_a, which is in turn given by Equation (4). Thus, one arrives at the self-consistency problem

$$\nu_a = \phi(\mu_a, \sigma_a) \,, \tag{8}$$

which is coupled across the populations due to Equation (6) and Equation (7).

Our toolbox provides two algorithms to solve Equation (8): (1) Integrating the auxiliary ordinary differential equation (ODE) $\dot{\nu}_a = -\nu_a + \phi(\mu_a, \sigma_a)$ with initial values $\nu_a(0) = \nu_{a,0}$ using `scipy.integrate.solve_ivp` (Virtanen et al., 2020) until it reaches a fixed point $\dot{\nu}_a = 0$, where Equation (8) holds by construction. (2) Minimizing the quadratic deviation $\sum_a \left[\nu_a - \phi(\mu_a, \sigma_a)\right]^2$, using the least squares (LSTSQ) solver `scipy.optimize.least_squares` (Virtanen et al., 2020) starting from an initial guess $\nu_{a,0}$. The ODE method is robust to changes in the initial values and hence a good first choice. However, it cannot find self-consistent solutions that correspond to an unstable fixed point of the auxiliary ODE (note that the stability of the auxiliary ODE does not indicate the stability of the solution). To this end, the LSTSQ method can be used. Its drawback is that it needs a good initial guess, because otherwise the found minimum might be a local one where the quadratic deviation does not vanish, $\sum_a \left[\nu_a - \phi(\mu_a, \sigma_a)\right]^2 > 0$, and which

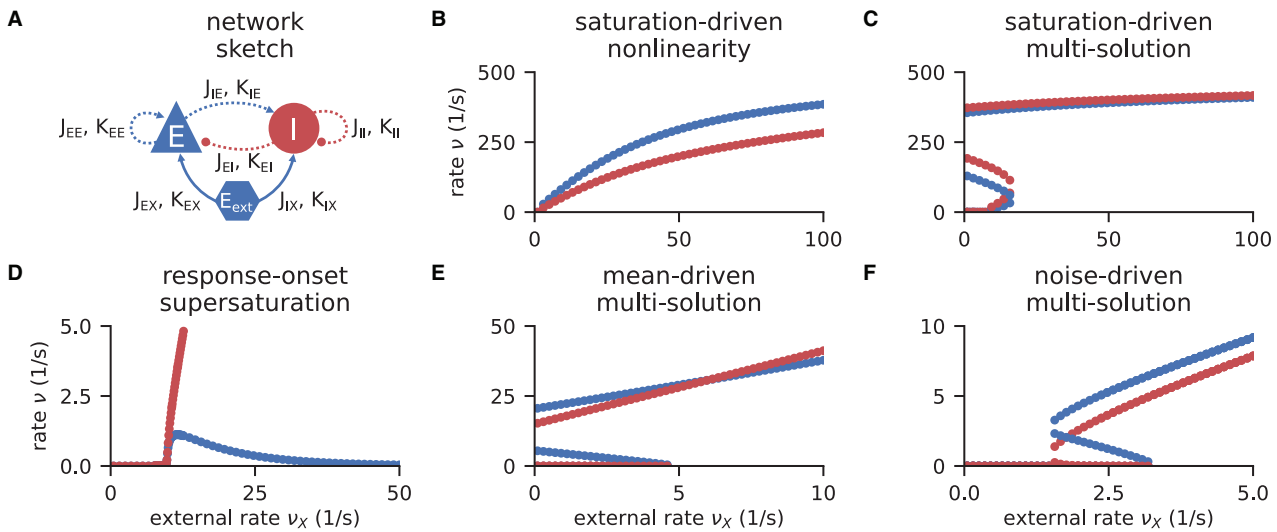

FIGURE 2 | Response nonlinearities in EI-networks. **(A)** Network diagram with nodes and edges according to the graphical notation proposed by Senk et al. (in press). **(B–F)** Firing rate of excitatory (blue) and inhibitory (red) population for varying external input rate ν_X. Specific choices for synaptic weights ($\boldsymbol{J}$, $\boldsymbol{J}_{\text{ext}}$) and in-degrees ($\boldsymbol{K}$, $\boldsymbol{K}_{\text{ext}}$) lead to five types of nonlinearities: **(B)** saturation-driven nonlinearity, **(C)** saturation-driven multi-solution, **(D)** response-onset supersaturation, **(E)** mean-driven multi-solution, and **(F)** noise-driven multi-solution. See Figure 8 in Sanzeni et al. (2020) for parameters.

accordingly does not correspond to a self-consistent solution, $\nu_a \neq \phi(\mu_a, \sigma_a)$. A prerequisite for both methods is a numerical solution of the integral in Equation (4); this is discussed in **Section A.1** in the **Appendix**.

The solutions of the self-consistency problem Equation (8) for varying ν_X and fixed $\boldsymbol{J}$, $\boldsymbol{J}_{\text{ext}}$, $\boldsymbol{K}$, and $\boldsymbol{K}_{\text{ext}}$ reveal the five types of response nonlinearities (**Figure 2**). Different response nonlinearities arise through specific choices of synaptic weights, $\boldsymbol{J}$ and $\boldsymbol{J}_{\text{ext}}$, and in-degrees, $\boldsymbol{K}$ and $\boldsymbol{K}_{\text{ext}}$, which suggests that already a simple EI-network possesses a rich capacity for nonlinear computations. Whenever possible, we use the ODE method and resort to the LSTSQ method only if the self-consistent solution corresponds to an unstable fixed point of the auxiliary ODE. Combining both methods, we can reproduce the first columns of Figure 8 in Sanzeni et al. (2020), where all five types of nonlinearities are presented.

In all cases, we chose appropriate initial values $\nu_{a,0}$ for either method. Note that an exploratory analysis is necessary if the stability properties of a network model are unknown, and potentially multiple fixed points are to be uncovered because there are, to the best of our knowledge, no systematic methods in $d > 1$ dimensions that provide all solutions of a nonlinear system of equations.

In **Listing 2**, we show a minimal example to produce the data shown in **Figure 2B**. After importing the function that solves the self-consistency Equation (8), we collect the neuron and network parameters in a dictionary. Then, we loop through different values for the external rate ν_X and determine the network rates using the ODE method, which is sufficient in this example. In **Listing 2** and to produce **Figure 2B**, we use the basic workflow because only one isolated tool of NNMT (`nnmt.lif.delta._firing_rates()`) is employed, which requires only a few parameters defining the simple EI-network.

```python
import numpy as np
from nnmt.lif.delta import _firing_rates

params = dict(
    # membrane and refractory time constants (in s)
    tau_m=20.*1e-3, tau_r=2.*1e-3,
    # relative reset and threshold potentials (in V)
    V_0_rel=10.*1e-3, V_th_rel=20.*1e-3,
    # recurrent and external weights (in V)
    J=np.array([[0.2, -1.6], [0.2, -1.4]])*1e-3,
    J_ext=np.array([0.2, 0.2])*1e-3,
    # recurrent and external in-degrees
    K=np.array([[400, 100], [400, 100]]),
    K_ext=np.array([1600, 800]),
    # set the method for the fixpoint finder
    fixpoint_method='ODE',
    # initial guess for the firing rate
    nu_0=(0, 0))

# determine self-consistent rates (in 1/s)
nu_ext = np.linspace(1, 100, 50) # external rates (in 1/s)
nu_E, nu_I = np.zeros_like(nu_ext), np.zeros_like(nu_ext)
for i, nu_X in enumerate(nu_ext):
    nu_E[i], nu_I[i] = _firing_rates(nu_ext=nu_X,
                                     **params)
```

Listing 2: Example script to produce the data shown in **Figure 2B** using the ODE method (initial value $\nu_{a,0} = 0$ for population $a \in \{E, I\}$).

3.2.2. Firing Rates of Microcircuit Model

Here we show how to use the model workflow to calculate the firing rates of the cortical microcircuit model by Potjans and Diesmann (2014). The circuit is a simplified point

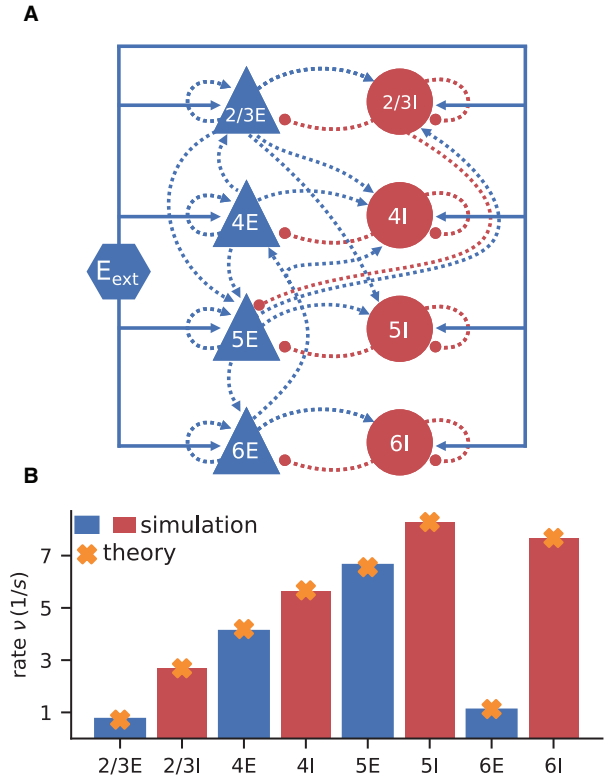

FIGURE 3 | Cortical microcircuit model by Potjans and Diesmann (2014). **(A)** Network diagram (only the strongest connections are shown as in Figure 1 of the original publication). Same notation as in **Figure 2A**. **(B)** Simulation and mean-field estimate for average population firing rates using the parameters from Bos et al. (2016).

neuron network model with biologically plausible parameters, which has been recently used in a number of other works: for example, to study network properties such as layer-dependent attentional processing (Wagatsuma et al., 2011), connectivity structure with respect to oscillations (Bos et al., 2016), and the effect of synaptic weight resolution on activity statistics (Dasbach, Tetzlaff, Diesmann, and Senk, 2021); to assess the performance of different simulator technologies such as neuromorphic hardware (van Albada et al., 2018) and GPUs (Knight and Nowotny, 2018; Golosio et al., 2021); to demonstrate forward-model prediction of local-field potentials from spiking activity (Hagen et al., 2016); and to serve as a building block for large-scale models (Schmidt et al., 2018).

The model consists of eight populations of LIF neurons, corresponding to the excitatory and inhibitory populations of four cortical layers: 2/3E, 2/3I, 4E, 4I, 5E, 5I, 6E, and 6I (see **Figure 3A**). It defines the number of neurons in each population, the number of connections between the populations, the single neuron properties, and the external input. Simulations show that the model yields realistic firing rates for the different populations as observed in particular in the healthy resting-state of early sensory cortex (Potjans and Diesmann, 2014, Table 6).

In contrast to the EI-network model investigated in Section 3.2.1, the neurons in the microcircuit model have exponentially shaped post-synaptic currents: Equation (2) is replaced by Fourcaud and Brunel (2002)

$$\tau_s R \frac{\mathrm{d}I_i}{\mathrm{d}t}(t) = -RI_i(t) + \tau_m \sum_j J_{ij} \sum_k \delta(t - t_{j,k} - d_{ij}), \quad (9)$$

with synaptic time constant τ_s. Note that J_{ij} is a measure in volts here. As discussed in Section 3.2.1, in mean-field theory the second term, representing the neuronal input, is approximated by Gaussian white noise. The additional synaptic filtering leads to the membrane potential (Equation 1) receiving colored noise input. Fourcaud and Brunel (2002) developed a method for calculating the firing rate for this synapse type. They have shown that, if the synaptic time constant τ_s is much smaller than the membrane time constant τ_m, the firing rate for LIF neurons with exponential synapses can be calculated using Equation (4) with shifted integration boundaries

$$\begin{aligned} \widetilde{V}_{\mathrm{cn},0}(\mu,\sigma) &= \widetilde{V}_0(\mu,\sigma) + \frac{\alpha}{2}\sqrt{\frac{\tau_s}{\tau_m}}, \\ \widetilde{V}_{\mathrm{cn,th}}(\mu,\sigma) &= \widetilde{V}_{\mathrm{th}}(\mu,\sigma) + \frac{\alpha}{2}\sqrt{\frac{\tau_s}{\tau_m}}, \end{aligned} \quad (10)$$

with the rescaled reset- and threshold-voltages from Equation (5) and $\alpha = \sqrt{2}\,|\zeta(1/2)| \approx 2.07$, where $\zeta(x)$ denotes the Riemann zeta function; the subscript cn stands for "colored noise".

The microcircuit has been implemented as an NNMT model (`nnmt.models.Microcircuit`). We here use the parameters of the circuit as published in Bos et al. (2016) which is slightly differently parameterized than the original model (see **Table A1** in the **Appendix**). The parameters of the model are specified in a `yaml` file, which uses Python-like indentation and a dictionary-style syntax. List elements are indicated by hyphens, and arrays can be defined as nested lists. Parameters with units can be defined by using the keys `val` and `unit`, whereas unitless variables can be defined without any keys. **Listing 3** shows an example of how some of the microcircuit network parameters used here are defined. Which parameters need to be provided in the yaml file depends on the model used and is indicated in their respective docstrings.

Once the parameters are defined, a microcircuit model is instantiated by passing the respective parameter file to the model constructor; the units are automatically converted to SI units. Then the firing rates are computed. For comparison, we finally load the simulated rates from Bos et al. (2016):

```
# create the network model using a network parameter yaml
# file
microcircuit = nnmt.models.Microcircuit(
    'network_params_microcircuit.yaml')
# calculate firing rates
firing_rates = nnmt.lif.exp.firing_rates(microcircuit)
# load simulated results
simulated_firing_rates = \
    nnmt.input_output.load_h5('Bos2016_rates.h5')['rates']
```

The simulated rates have been obtained by a numerical network simulation (for simulation details see Bos et al., 2016) in which

```yaml
# membrane time constant
tau_m:
  val: 10.0
  unit: ms

# neuron numbers
N:
  - 20683
  - 5834
  - 21915
```

Listing 3: Some microcircuit network parameters defined in a yaml file. A dictionary-like structure with the keys `val` (value) and `unit` is used to define the membrane time constant, which is the same across all populations. The numbers of neurons in each population are defined as a list. Only the numbers for the first three populations are displayed.

the neuron populations are connected according to the model's original connectivity rule: "random, fixed total number with multapses (autapses prohibited)", see Senk et al. (in press) as a reference for connectivity concepts. The term *multapses* refers to multiple connections between the same pair of neurons and *autapses* are self-connections; with this connectivity rule multapses can occur in a network realization but autapses are not allowed. For simplicity, the theoretical predictions assume a connectivity with a fixed in-degree for each neuron. Dasbach et al. (2021) show that simulated spike activity data of networks with these two different connectivity rules are characterized by differently shaped rate distributions ("reference" in their Figures 3d and 4d). In addition, the weights in the simulation are normally distributed while the theory replaces each distribution by its mean; this corresponds to the case $N_{\text{bins}} = 1$ in Dasbach et al. (2021). Nevertheless, our mean-field theoretical estimate of the average population firing rates is in good agreement with the simulated rates (**Figure 3B**).

3.3. Dynamical Quantities

3.3.1. Transfer Function

One of the most important dynamical properties of a neuronal network is how it reacts to external input. A systematic way to study the network response is to apply an oscillatory external input current leading to a periodically modulated mean input $\mu(t) = \mu + \delta\mu \,\mathrm{Re}\left(\mathrm{e}^{\mathrm{i}\omega t}\right)$ (cf. Equation 6), with fixed frequency ω, phase, and amplitude $\delta\mu$, and observe the emerging frequency, phase, and amplitude of the output. If the amplitude of the external input is small compared to the stationary input, the network responds in a linear fashion: it only modifies phase and amplitude, while the output frequency equals the input frequency. This relationship is captured by the input-output transfer function $N(\omega)$ (Brunel and Hakim, 1999; Brunel et al., 2001; Lindner and Schimansky-Geier, 2001), which describes the frequency-dependent modulation of the output firing rate of a neuron population

$$\nu(t) = \nu + \mathrm{Re}\Big(N(\omega)\,\delta\mu\,\mathrm{e}^{\mathrm{i}\omega t}\Big).$$

Note that in this section we only study the linear response to a modulation of the mean input, although in general, a modulation of the noise intensity (Equation 7) can also be included (Lindner and Schimansky-Geier, 2001; Schuecker et al., 2015). The transfer function $N(\omega)$ is a complex function: Its absolute value describes the relative modulation of the firing rate. Its phase, the angle relative to the real axis, describes the phase shift that occurs between input and output. We denote the transfer function for a network of LIF neurons with instantaneous synapses in linear-response approximation as

$$N(\omega) = \frac{\sqrt{2}\nu}{\sigma}\,\frac{1}{1+\mathrm{i}\omega\tau_{\mathrm{m}}}\,\frac{\Phi'_\omega\big|_{\sqrt{2}\widetilde{V}_0}^{\sqrt{2}\widetilde{V}_{\mathrm{th}}}}{\Phi_\omega\big|_{\sqrt{2}\widetilde{V}_0}^{\sqrt{2}\widetilde{V}_{\mathrm{th}}}}, \tag{11}$$

with the rescaled reset- and threshold-voltages $\widetilde{V}_0$ and $\widetilde{V}_{\mathrm{th}}$ as defined in Equation (5) and $\Phi_\omega(x) = e^{\frac{x^2}{4}}\,U\left(\mathrm{i}\omega\tau_{\mathrm{m}} - \frac{1}{2}, x\right)$ using the parabolic cylinder functions $U\left(\mathrm{i}\omega\tau_{\mathrm{m}} - \frac{1}{2}, x\right)$ as defined in (Abramowitz and Stegun, 1974, Section 19.3) and (Olver et al., 2021, Section 12.2). Φ'_ω denotes the first derivative by x. A comparison of our notation and the transfer function given in Schuecker et al. (2015, Equation 29) can be found in **Section A.2.1** in the **Appendix**.

For a neuronal network of LIF neurons with exponentially shaped post-synaptic currents, introduced in Section 3.2.2, Schuecker et al. (2014, 2015) show that an analytical approximation of the transfer function can be obtained by a shift of integration boundaries, akin to Equation (10):

$$N_{\mathrm{cn}}(\omega) = \frac{\sqrt{2}\nu}{\sigma}\,\frac{1}{1+\mathrm{i}\omega\tau_{\mathrm{m}}}\,\frac{\Phi'_\omega\big|_{\sqrt{2}\widetilde{V}_{\mathrm{cn,0}}}^{\sqrt{2}\widetilde{V}_{\mathrm{cn,th}}}}{\Phi_\omega\big|_{\sqrt{2}\widetilde{V}_{\mathrm{cn,0}}}^{\sqrt{2}\widetilde{V}_{\mathrm{cn,th}}}}. \tag{12}$$

To take into account the effect of the synaptic dynamics, we include an additional low-pass filter:

$$N_{\mathrm{cn,s}}(\omega) = N_{\mathrm{cn}}(\omega)\,\frac{1}{1+\mathrm{i}\omega\tau_{\mathrm{s}}}. \tag{13}$$

If the synaptic time constant is much smaller than the membrane time constant ($\tau_{\mathrm{s}} \ll \tau_{\mathrm{m}}$), an equivalent expression for the transfer function is obtained by a Taylor expansion around the original boundaries (cf. Schuecker et al. 2015, Equation 30). The toolbox implements both variants and offers choosing between them by setting the argument `method` of `nnmt.lif.exp.transfer_function` to either `shift` or `taylor`.

Here, we demonstrate how to calculate the analytical "shift version" of the transfer function for different means and noise intensities of the input current (see **Figure 4**) and thereby reproduce Figure 4 in Schuecker et al. (2015).

The crucial parts for producing **Figure 4** using NNMT are shown in **Listing 4** for one example combination of mean and noise intensity of the input current. Instead of using the model workflow with `nnmt.lif.exp.transfer_function`, we here employ the basic workflow, using

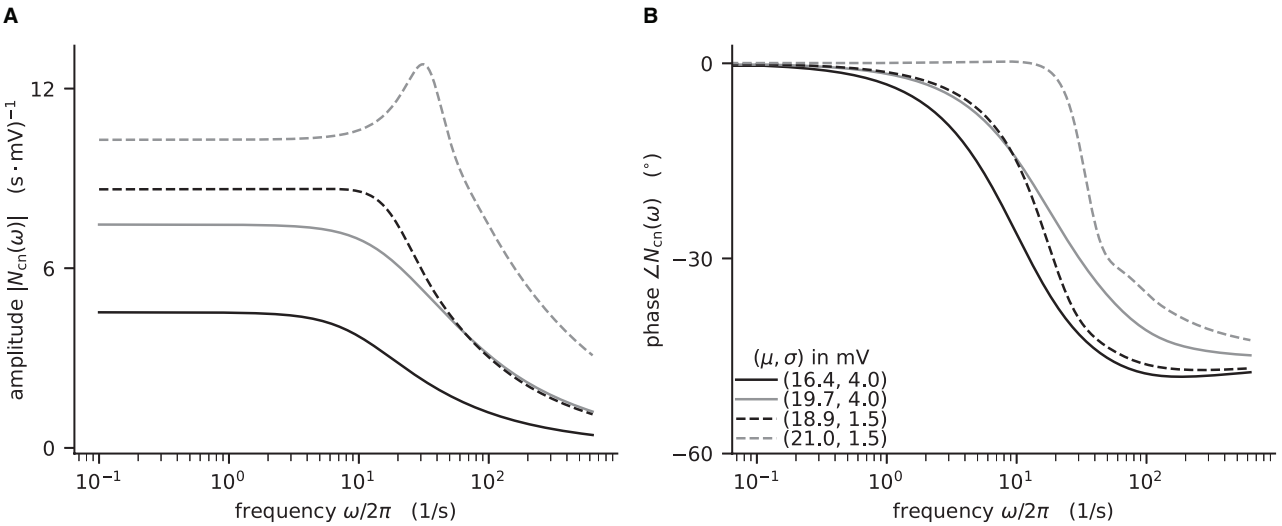

FIGURE 4 | Colored-noise transfer function N_{cn} of LIF model in different regimes. **(A)** Absolute value and **(B)** phase of the "shift" version of the transfer function as a function of the log-scaled frequency. Neuron parameters are set to $V_{th} = 20$ mV, $V_0 = 15$ mv, $\tau_m = 20$ ms, and $\tau_s = 0.5$ ms. For given noise intensities of input current, $\sigma = 4$ mV (solid line) and $\sigma = 1.5$ mV (dashed line), the mean input μ is chosen such that firing rates $\nu = 10$ Hz (black) and $\nu = 30$ Hz (gray) are obtained.

`nnmt.lif.exp._transfer_function` directly. This allows changing the mean input and its noise intensity independently of a network model's structure, but requires two additional steps: First, the necessary parameters are loaded from a `yaml` file, converted to SI units and then stripped off the units using the utility function `nnmt.utils._convert_to_si_and_strip_units`. Second, the analysis frequencies are defined manually. In this example we choose logarithmically spaced frequencies, as we want to plot the results on a log-scale. Finally, the complex-valued transfer function is calculated and then split into its absolute value and phase. **Figure 4** shows that the transfer function acts as a low-pass filter that suppresses the amplitude of high frequency activity, introduces a phase lag, and can lead to resonance phenomena for certain configurations of mean input current and noise intensity.

The replication of the results from Schuecker et al. (2015) outlined here is also used in the integration tests of the toolbox. Note that the implemented analytical form of the transfer function by Schuecker et al. (2015) is an approximation for low frequencies, and deviations from a simulated ground truth are expected for higher frequencies ($\omega/2\pi \gtrsim 100$ Hz at the given parameters).

3.3.2. Power Spectrum

Another frequently studied dynamical property is the power spectrum, which describes how the power of a signal is distributed across its different frequency components, revealing oscillations of the population activity. The power is the Fourier transformed auto-correlation of the population activities (c.f. Bos et al. 2016, Equations 16-18). Linear response theory on top of a mean-field approximation, allows computing the power, dependent on the network architecture, the stationary firing rates, and the neurons' transfer function (Bos et al., 2016). The corresponding analytical expression for the power spectra of population a at angular frequency ω is given by the diagonal elements of the correlation matrix

$$\begin{aligned} P_a(\omega) &= C_{aa}(\omega) \\ &= \left[\left(\mathbf{1} - \widetilde{\mathbf{M}}_{\mathrm{d}}(\omega)\right)^{-1} \operatorname{diag}\left(\boldsymbol{\nu} \oslash \boldsymbol{n}\right) \left(\mathbf{1} - \widetilde{\mathbf{M}}_{\mathrm{d}}(-\omega)\right)^{-\mathrm{T}} \right]_{aa}, \end{aligned} \tag{14}$$

with $\oslash$ denoting the elementwise (Hadamard) division, the effective connectivity matrix $\widetilde{\boldsymbol{M}}_{\mathrm{d}}(\omega) = \tau_{\mathrm{m}} \boldsymbol{N}_{\mathrm{cn,s}}(\omega) \cdot \boldsymbol{J} \odot \boldsymbol{K} \odot \boldsymbol{D}(\omega)$, where the dot denotes the scalar product, while $\odot$ denotes the elementwise (Hadamard) product, the mean population firing rates $\boldsymbol{\nu}$, and the numbers of neurons in each population $\boldsymbol{n}$. The effective connectivity combines the static, anatomical connectivity $\boldsymbol{J} \odot \boldsymbol{K}$, represented by synaptic weight matrix $\boldsymbol{J}$ and in-degree matrix $\boldsymbol{K}$, and dynamical quantities, represented by the transfer functions $N_{\mathrm{cn,s},a}(\omega)$ (Equation (13)), and the contribution of the delays in (Equation 13), represented by their Fourier transformed distributions $D_{ab}(\omega)$ (cf. Bos et al. 2016, Equations 14, 15).

The modular structure in combination with the model workflow of this toolbox permits a step-by-step calculation of the power spectra, as shown in **Listing 5**. The inherent structure of the theory is emphasized in these steps: After instantiating the network model class with given network parameters, we determine the working point, which characterizes the statistics of the model's stationary dynamics. It is defined by the population firing rates, the mean, and the standard deviation of the input to a neuron of the respective population. This is necessary for determining the transfer functions. The calculation of the delay distribution matrix is then required for calculating the effective connectivity and to finally get an estimate of the power spectra.

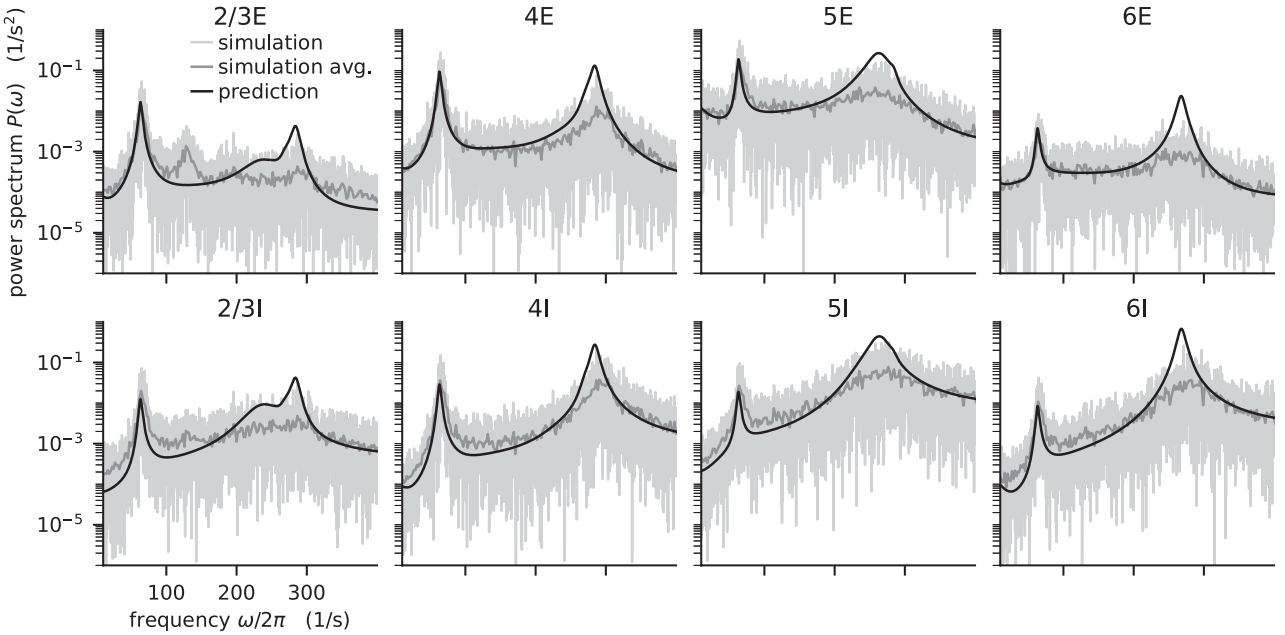

FIGURE 5 | Power spectra of the population spiking activity in the adapted cortical microcircuit from Bos et al. (2016). The spiking activity of each population in a 10 s simulation of the model is binned with 1 ms resolution and the power spectrum of the resulting histogram is calculated by a fast Fourier transform (FFT; light gray curves). In addition, the simulation is split into 500 ms windows, the power spectrum calculated for each window and averaged across windows (gray curves). Black curves correspond to analytical prediction obtained with NNMT as described in **Listing 5**. The panels show the spectra for the excitatory (top) and inhibitory (bottom) populations within each layer of the microcircuit.

Figure 5 reproduces Figure 1E in Bos et al. (2016) and shows the spectra for each population of the adjusted version (see **Table A1** in the **Appendix**) of the microcircuit model.

The numerical predictions obtained from the toolbox largely coincide with simulated data taken from the original publication (Bos et al., 2016) and reveal dominant oscillations of the population activities in the low-γ range around 63 Hz. Furthermore, faster oscillations with peak power around 300 Hz are predicted with higher magnitudes in the inhibitory populations 4I, 5I, and 6I.

The deviation between predicted and simulated power spectra seen at $\sim$ 130 Hz in population 2/3E could be a harmonic of the correctly predicted, prominent 63 Hz peak; a non-linear effect not captured in linear response theory. Furthermore, the systematic overestimation of the power spectrum at large frequencies is explained by the limited validity of the analytical approximation of the transfer function for high frequencies.

3.3.3. Sensitivity Measure

The power spectra shown in the previous section exhibit prominent peaks at certain frequencies, which indicate oscillatory activity. Naturally, this begs the question: which mechanism causes these oscillations? Bos et al. (2016) expose the crucial role that the microcircuit's connectivity plays in shaping the power spectra of this network model. They have developed a method called *sensitivity measure* to directly relate the influence of the anatomical connections, especially the in-degree matrix, on the power spectra.

The power spectrum of the a-th population $P_a(\omega)$ receives a contribution from each eigenvalue λ_b of the effective connectivity matrix, $P_a(\omega) \propto 1/\left(1-\lambda_b(\omega)\right)^2$. Such a contribution consequently diverges as the complex-valued λ_b approaches $1 + 0\mathrm{i}$ in the complex plane, which is referred to as the point of instability. This relation can be derived by replacing the effective connectivity matrix $\widetilde{\mathbf{M}}_\mathrm{d}(\omega)$ in Equation (14) by its eigendecomposition. The sensitivity measure leverages this relationship and evaluates how a change in the in-degree matrix affects the eigenvalues of the effective connectivity and thus indirectly the power spectrum. Bos et al. (2016) introduce a small perturbation α_{cd} of the in-degree matrix, which allows writing the effective connectivity matrix as $\widehat{M}_{ab}(\omega) = (1+\alpha_{cd}\delta_{ca}\delta_{db})\,\widetilde{M}_{ab}(\omega)$, where we dropped the delay subscript d. The sensitivity measure $Z_{b,cd}(\omega)$ describes how the b-th eigenvalue of the effective connectivity matrix varies when the cd-th element of the in-degree matrix is changed

$$Z_{b,cd}(\omega) = \left.\frac{\partial\lambda_b(\omega)}{\partial\alpha_{cd}}\right|_{\alpha_{cd}=0} = \frac{v_{b,c}\widetilde{M}_{cd}u_{b,d}}{\mathbf{v}_b^\mathrm{T}\cdot\mathbf{u}_b}\,, \tag{15}$$

where $\frac{\partial\lambda_b(\omega)}{\partial\alpha_{cd}}$ is the partial derivative of the eigenvalue with respect to a change in connectivity, $\mathbf{v}_b^\mathrm{T}$ and $\mathbf{u}_b$ are the left and right eigenvectors of $\widetilde{\boldsymbol{M}}$ corresponding to eigenvalue $\lambda_b(\omega)$.

The complex sensitivity measure can be understood in terms of two components: $\mathbf{Z}_b^\mathrm{amp}$ is the projection of the matrix $\mathbf{Z}_b$ onto the direction in the complex plane defined by $1-\lambda_b(\omega)$;

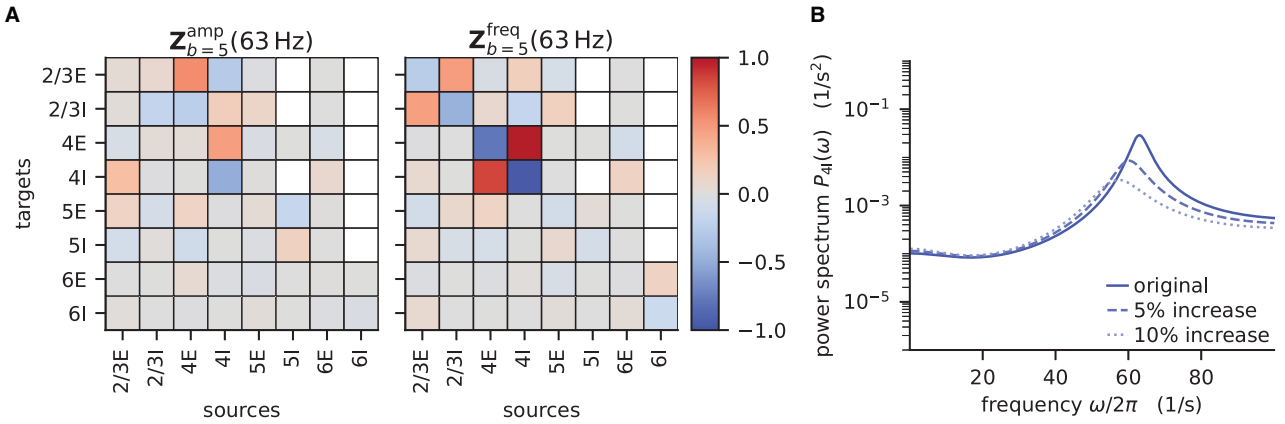

FIGURE 6 | Sensitivity measure at low-γ frequency and corresponding power spectrum of microcircuit with adjusted connectivity. **(A)** Sensitivity measure of one eigenmode of the effective connectivity relevant for low-γ oscillations. The sensitivity measure for this mode is evaluated at the frequency where the corresponding eigenvalue is closest to the point of instability $1 + 0i$ in complex plane. $\boldsymbol{Z}^{amp}_{b}(\omega)$ (left subpanel) visualizes the influence of a perturbation of a connection on the peak amplitude of the power spectrum. $\boldsymbol{Z}^{freq}_{b}(\omega)$ (right subpanel) shows the impact on the peak frequency. Non-existent connections are masked white. **(B)** Mean-field prediction of power spectrum of population 4I with original connectivity parameters (solid line), 5% increase (dashed line) and 10% increase (dotted line) in connections $K_{4I\to 4I}$. The increase in inhibitory input to population 4I was counteracted by an increase of the excitatory external input $K_{ext\to 4I}$ to maintain the working point.

it describes how, when the in-degree matrix is perturbed, the complex-valued $\lambda_b(\omega)$ moves toward or away from the instability $1 + 0i$, and consequently how the amplitude of the power spectrum at frequency ω increases or decreases. $\mathbf{Z}^{freq}_{b}$ is the projection onto the perpendicular direction and thus describes how the peak frequency of the power spectrum changes with the perturbation of the in-degree matrix. For a visualization of these projections, refer to Figure 5B in Bos et al. (2016).

The toolbox makes this intricate measure accessible by supplying two tools: After computing the required working point, transfer function, and delay distribution, the tool `nnmt.lif.exp.sensitivity_measure` computes the sensitivity measure at a given frequency for one specific eigenvalue. By default, this is the eigenvalue which is closest to the instability $1 + 0i$. To perform the computation, we just need to add one line to **Listing 5**:

```
sensitivity_dict = nnmt.lif.exp.sensitivity_measure(
    microcircuit, frequency)
```

The result is returned in form of a dictionary that contains the sensitivity measure and its projections. The tool `nnmt.lif.exp.sensitivity_measure_all_eigenmodes` wraps that basic function and calculates the sensitivity measure for all eigenvalues at the frequency for which each eigenvalue is closest to instability.

According to the original publication (Bos et al., 2016), the peak around 63 Hz has contributions from one eigenvalue of the effective connectivity matrix. **Figure 6** shows the projections of the sensitivity measure at the frequency for which this eigenvalue is closest to the instability, as illustrated in Figure 4 of Bos et al. (2016). The sensitivity measure returns one value for each connection between populations in the network model. For $\mathbf{Z}^{amp}_{b}$ a negative value indicates that increasing the in-degrees of a specific connection causes the amplitude of the power spectrum at the evaluated frequency to drop. If the value is positive, the amplitude is predicted to grow as the in-degrees increase. Similarly, for positive $\mathbf{Z}^{freq}_{b}$ the frequency of the peak in the power spectrum shifts toward higher values as in-degrees increase, and vice versa. The main finding in this analysis is that the low-γ peak seems to be affected by excitatory-inhibitory loops in layer 2/3 and layer 4.

To decrease the low-γ peak in the power spectrum, one could therefore increase the 4I to 4I connections (cp. **Figure 6A**):

```
# 5 percent increase
K_new = microcircuit.network_params['K'].copy()
K_new[3,3] = 1001 # originally 953
K_ext_new = microcircuit.network_params['K_ext'].copy()
K_ext_new[3] = 2034 # originally 1900
microcircuit_new = microcircuit.change_parameters(
    {'K': K_new, 'K_ext': K_ext_new})
```

and calculate the power spectrum as in **Listing 5** again to validate the change. Note that a change in connectivity leads to a shift in the working point. We are interested in the impact of the modified connectivity on the fluctuation dynamics at the same working point and thus need to counteract the change in connectivity by adjusting the external input. In the chosen example this is ensured by satisfying $J_{4I\to 4I}\Delta K_{4I\to 4I}\nu_{4I} = -J_{ext\to 4I}\Delta K_{ext\to 4I}\nu_{ext}$, which yields $\Delta K_{ext\to 4I} = -\frac{J_{4I\to 4I}\Delta K_{4I\to 4I}\nu_{4I}}{J_{ext\to 4I}\nu_{ext}}$.

If several eigenvalues of the effective connectivity matrix influence the power spectra in the same frequency range, adjustments of the connectivity are more involved. This is because a change in connectivity would inevitably affect all eigenvalues simultaneously. Further care has to be taken because the sensitivity measure is subject to the same constraints as the current implementation of the transfer function, which is only valid for low frequencies and enters the sensitivity measure directly.

```python
# load parameters in custom units
params = nnmt.input_output.load_val_unit_dict_from_yaml(
    'Schuecker2015_parameters.yaml')

# convert parameters to SI units
nnmt.utils._convert_to_si_and_strip_units(params)

# define the analysis frequencies
frequencies = np.logspace(
    params['f_start_exponent'],
    params['f_end_exponent'],
    params['n_freqs'])
# add the zero frequency
frequencies = np.insert(frequencies, 0, 0.0)
omegas = 2 * np.pi * frequencies

# extract necessary parameters from params dictionary
mean_input = params['mean_input']
...  # here we leave out similar statements

# calculate the transfer function
transfer_function = nnmt.lif.exp._transfer_function(
    mu, sigma,
    tau_m, tau_s, tau_r,
    V_th_rel, V_0_rel,
    omegas,
    method='shift',
    synaptic_filter=False)

# calculate properties plotted in Schuecker et al. (2015)
absolute_value = np.abs(transfer_function)
phase = np.angle(transfer_function) / 2 / np.pi * 360
```

Listing 4: Example script for computing a transfer function shown in **Figure 4** using the method of shifted integration boundaries.

```python
# create network model microcircuit
microcircuit = nnmt.models.Microcircuit(
    network_params='Bos2016_network_params.yaml',
    analysis_params='Bos2016_analysis_params.yaml')

# calculate working point for exponentially shaped post-
    synaptic currents
nnmt.lif.exp.working_point(microcircuit, method='taylor')
# calculate the transfer function
nnmt.lif.exp.transfer_function(microcircuit,
    method='taylor')
# calculate the delay distribution matrix
nnmt.network_properties.delay_dist_matrix(microcircuit)
# calculate the effective connectivity matrix
nnmt.lif.exp.effective_connectivity(microcircuit)
# calculate the power spectra
power_spectra = nnmt.lif.exp.power_spectra(microcircuit)
```

Listing 5: Example script to produce the theoretical prediction (black lines) shown in **Figure 5B**.

3.4. Fitting Spiking to Rate Model and Predicting Pattern Formation

If the neurons of a network are spatially organized and connected according to a distance-dependent profile, the spiking activity may exhibit pattern formation in space and time, including wave-like phenomena. Senk et al. (2020) set out to scrutinize the non-trivial relationship between the parameters of such a network model and the emerging activity patterns. The model they use is a two-population network of excitatory E and inhibitory I spiking neurons, illustrated in **Figure 7**. All neurons are of type LIF with exponentially shaped post-synaptic currents. The neuron populations are recurrently connected to each other and themselves and they receive additional external excitatory E_{ext} and inhibitory I_{ext} Poisson spike input of adjustable rate as shown in **Figure 7A**. The spatial arrangement of neurons on a ring is illustrated in **Figure 7B** and the boxcar-shaped connectivity profiles in **Figure 7C**.

In the following, we consider a mean-field approximation of the spiking model with spatial averaging, that is a time and space continuous approximation of the discrete model as derived in Senk et al. (2020, Section E. Linearization of spiking network model). We demonstrate three methods used in the original study: First, Section 3.4.1 explains how a model can be brought to a defined state characterized by its working point. The working point is given by the mean μ and noise intensity σ of the input to a neuron, which are both quantities derived from network parameters and require the calculation of the firing rates. With the spiking model in that defined state, Section 3.4.2 then maps its transfer function to the one of a rate model. Section 3.4.3 finally shows that this working-point dependent rate model allows for an analytical linear stability analysis of the network accounting for its spatial structure. This analysis can reveal transitions to spatial and temporal oscillatory states which, when mapped back

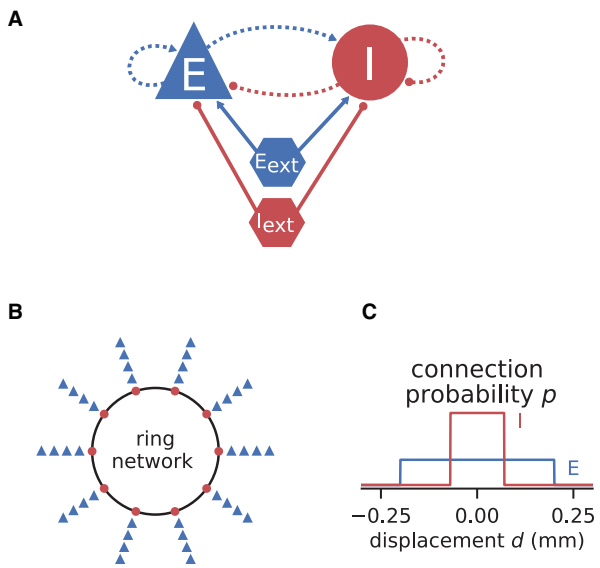

FIGURE 7 | Illustrations of spiking network model by Senk et al. (2020). **(A)** Excitatory and inhibitory neuronal populations randomly connected with fixed in-degree and multapses allowed (autapses prohibited). External excitatory and inhibitory Poisson drive to all neurons. Same notation as in **Figure 2A**. **(B)** One inhibitory and four excitatory neurons per grid position on a one-dimensional domain with periodic boundary conditions (ring network). **(C)** Normalized, boxcar-shaped connection probability with wider excitation than inhibition; the grid spacing is here 10^{-3} mm. For model details and parameters, see Tables II–IV of Senk et al. (2020); the specific values given in the caption of their Figure 6 are used throughout here.

to the parameters of the spiking model, may manifest in distinct patterns of simulated spiking activity after a startup transient.

3.4.1. Setting the Working Point by Changing Network Parameters

With network and analysis parameters predefined in `yaml` files, we set up a network model using the example model class `Basic`:

```
space_model = nnmt.models.Basic(
    network_params='Senk2020_network_params.yaml',
    analysis_params='Senk2020_analysis_params.yaml')
```

Upon initialization the given parameters are automatically converted into the format used by NNMT's tools. For instance, relative spike reset and threshold potentials are derived from the absolute values, connection strengths in units of volt are computed from the post-synaptic current amplitudes in ampere, and all values are scaled to SI units.

We aim to bring the network to a defined state by fixing the working point but also want to explore if the procedure of fitting the transfer function still works for different network states. For a parameter space exploration, we use a method to change parameters provided by the model class and scan through a number of different working points of the network. To obtain the required input for a target working point, we adjust the external excitatory and inhibitory firing rates accordingly; NNMT uses a vectorized version of the equations given in Senk et al. (2020, Appendix F: Fixing the working point) to calculate the external rates needed:

```
# relative to spike threshold (in V)
mu = 10. * 1e-3; sigma = 10. * 1e-3
nu_ext = nnmt.lif.exp.external_rates_for_fixed_input(
    space_model, mu_set=mu, sigma_set=sigma)
space_model = space_model.change_parameters(
    changed_network_params={'nu_ext': nu_ext})
```

The implementation uses only one excitatory and one inhibitory Poisson source to represent the external input rates which typically originate from a large number of external source neurons. These two external sources are connected to the network with the same relative inhibition g as used for the internal connections. The resulting external rates for different choices of (μ, σ) are color-coded in the first two plots of **Figure 8A**. The third plot shows the corresponding firing rates of the neurons, which are stored in the results of the model instance when computing the working point explicitly:

```
nnmt.lif.exp.working_point(space_model)
```

Although the external rates are substantially higher than the firing rates, since a neuron is recurrently connected to hundreds of neurons, the total external and recurrent inputs are of the same order.

3.4.2. Parameter Mapping by Fitting the Transfer Function

We map the parameters of the spiking model to a corresponding rate model by, first, computing the transfer function $\boldsymbol{N}_{\mathrm{cn,s}}$ given in Equation (13) of the spiking model, and second, fitting the simpler transfer function of the rate model, for details see Senk et al. (2020, Section F. Comparison of neural-field and spiking models). The dynamics of the rate model can be written as a differential equation for the linearized activity r_a with populations $a, b \in \{\mathrm{E}, \mathrm{I}\}$:

$$\tau \frac{\mathrm{d}}{\mathrm{d}t} r_a(t) = -r_a(t) + \sum_b w_b r_b(t-d) \tag{16}$$

with the delay d; τ is the time constant and w_b are the unitless weights that only depend on the presynaptic population. The transfer function is just the one of a low-pass filter, $N_{\mathrm{LP}} = 1/(1+\lambda\tau)$, where λ is the frequency in Laplace domain. The tool to fit the transfer function requires that the actual transfer function $\boldsymbol{N}_{\mathrm{cn,s}}$ has been computed beforehand and fits $N_{\mathrm{LP}}\boldsymbol{w}$ to $\tau_{\mathrm{m}}\boldsymbol{N}_{\mathrm{cn,s}} \cdot \boldsymbol{J} \odot \boldsymbol{K}$ for the same frequencies together with τ, $\boldsymbol{w}$, and the combined fit error η:

```
nnmt.lif.exp.transfer_function(space_model)
nnmt.lif.exp.fit_transfer_function(space_model)
```

The absolute value of the transfer function is fitted with non-linear least-squares using the solver `scipy.optimize.curve_fit`. **Figure 8B** illustrates the amplitude and phase of the transfer function and its fit for a few (μ, σ) combinations. The plots of **Figure 8C** show the fitted time constants, the fitted excitatory weight, and the combined fit error. The inhibitory weight is proportional to the excitatory one in the same way as the post-synaptic current amplitudes.

3.4.3. Linear Stability Analysis of Spatially Structured Model With Delay

Sections 3.4.1 and 3.4.2 considered a mean-field approximation of the spiking model without taking space into account. In the following, we assume a spatial averaging of the discrete network depicted in **Figure 7** and introduce the spatial connectivity profiles $p_a(x)$. Changing Equation (16) to the integro-differential equation

$$\tau \frac{\partial}{\partial t} r_a(x,t) = -r_a(x,t) + \sum_b w_b \int_{-\infty}^{\infty} p_b(x-y) r_b(y, t-d)\,\mathrm{d}y \tag{17}$$

yields a neural field model defined in continuous space x. This model lends itself to analytical linear stability analysis, as described in detail in Senk et al. (2020, Section A. Linear stability analysis of a neural-field model). In brief, we analyze the stability of a fixed-point solution to this differential equation system which, together with parameter continuation methods and bifurcation analysis, allows us to determine points in parameter space where transitions from homogeneous steady states to oscillatory states can occur. These transitions are given as a function of a bifurcation parameter, here the constant delay d, which is the same for all connections. The complex-valued, temporal eigenvalue λ of the linearized time-delay system is an indicator for the system's overall stability and can serve as a predictor for temporal oscillations, whereas the spatial oscillations are characterized by the real-valued wave number k. Solutions that relate λ and k with the model parameters are given by a characteristic equation, which in our case reads (Senk et al., 2020, Equation 7):

$$\lambda_B(k) = -\frac{1}{\tau} + \frac{1}{d} W_B\left(c(k)\frac{d}{\tau}\mathrm{e}^{\frac{d}{\tau}}\right), \tag{18}$$

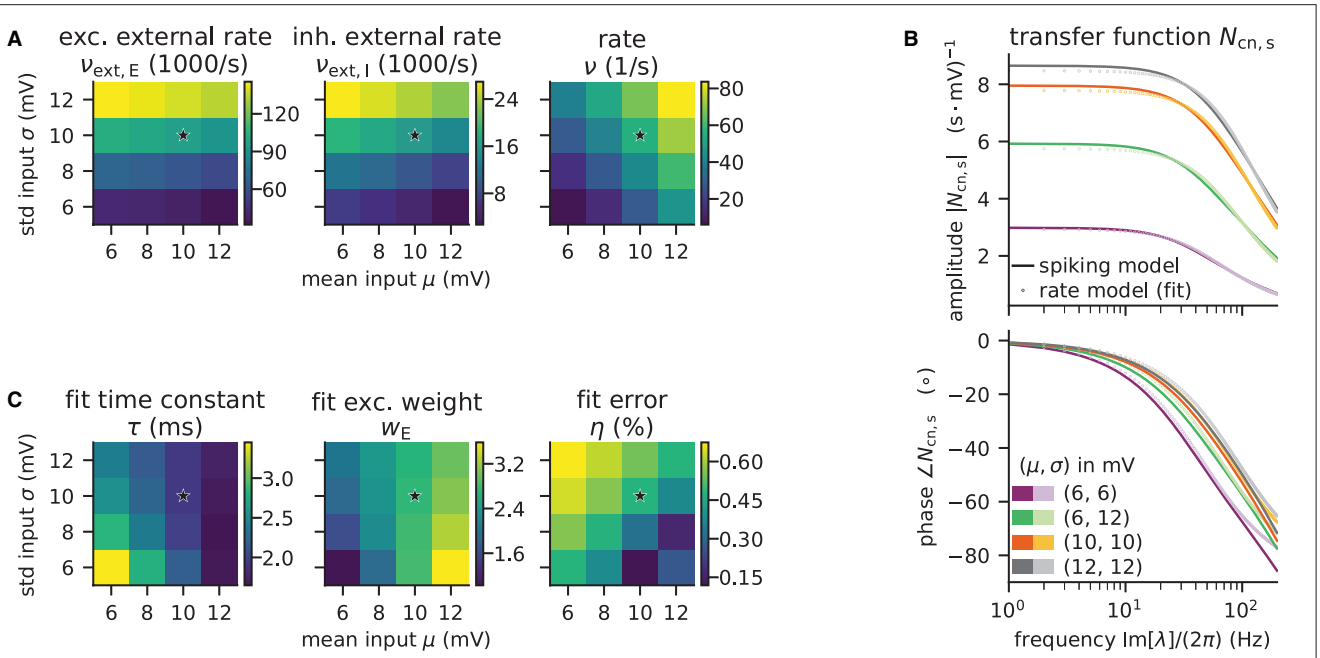

FIGURE 8 | Network parameters and mean-field results from scanning through different working points. Working point (μ, σ) combines mean input μ and noise intensity of input σ. **(A)** External excitatory $\nu_{\text{ext,E}}$ and inhibitory $\nu_{\text{ext,I}}$ Poisson rates required to set (μ, σ) and resulting firing rates ν. **(B)** Transfer function $N_{\text{cn,s}}$ of spiking model and fitted rate-model approximation with low-pass filter for selected (μ, σ) (top: amplitude, bottom: phase). **(C)** Fit results (time constants τ and excitatory weights w_{E}) and fit errors η. The inhibitory weights are $w_{\text{I}} = -gw_{\text{E}}$ with $g = 5$. Star marker in panels **(A)** and **(C)** denotes target working point (10, 10) mV. Similar displays as in Senk et al. (2020, Figure 5).

with the time constant of the rate model τ, the multi-valued Lambert W_B function[3] on branch B (Corless et al., 1996), and the effective connectivity profile $c\left(k\right)$, which combines the weights w_b and the Fourier transforms of the spatial connectivity profiles. Note that the approach generalizes from the boxcar-shaped profiles used here to any symmetric probability density function. NNMT provides an implementation to solve this characteristic equation in its `linear_stability` module using the `spatial` module for the profile:

```
import nnmt.spatial as spatial
import nnmt.linear_stability as linstab

connectivity = (
    W_rate * spatial._ft_spatial_profile_boxcar(
        k_wavenumber,
        space_model.network_params['width']))

eigenvalue = (
    linstab._solve_chareq_lambertw_constant_delay(
        branch_nr, tau_rate,
        space_model.network_params['delay'],
        connectivity))
```

Figure 9A shows that the computed eigenvalues come for the given network parameters in complex conjugate pairs. The branch with the largest real part is the principal branch ($B = 0$). Temporal oscillations are expected to occur if the real part of the eigenvalue on the principal branch becomes positive; the oscillation frequency can then be read off the imaginary part of that eigenvalue. In this example, the largest eigenvalue λ^* on the principal branch has a real part that is just above zero. There exists a supercritical Hopf bifurcation and the delay as the bifurcation parameter is chosen large enough such that the model is just beyond the bifurcation point separating the stable from the instable state. The respective wave number k^* is positive, which indicates spatial oscillations as well. The oscillations in both time and space predicted for the rate model imply that the activity of the corresponding spiking model might exhibit wave trains, i.e., temporally and spatially periodic patterns. The predicted propagation speed of the wave trains is given by the phase velocity $\text{Im}\left[\lambda^*\right]/k^*$.

To determine whether the results obtained with the rate model are transferable to the spiking model, **Figure 9B** interpolates the analytical solutions of the rate model [$\alpha = 0$, evaluating Equation (18)] to solutions of the spiking model ($\alpha = 1$, accounting for the transfer function $\boldsymbol{N}_{\text{cn,s}}$), which can only be computed numerically. Thus, the parameter α interpolates between the characteristic equations of these two models which primarily differ in their transfer function; for details see Senk et al. (2020, Section F.2 Linear interpolation between the transfer functions). Since the eigenvalues estimated this way show only little differences between rate and spiking model, we conclude that predictions from the rate model should hold also for the spiking model in the parameter regime tested. Following the

[3]The Lambert W_B function is defined as $z = W_B\left(z\right)\mathrm{e}^{W_B(z)}$ for $z \in \mathbb{C}$ and has infinitely many solutions, numbered by the branches B.

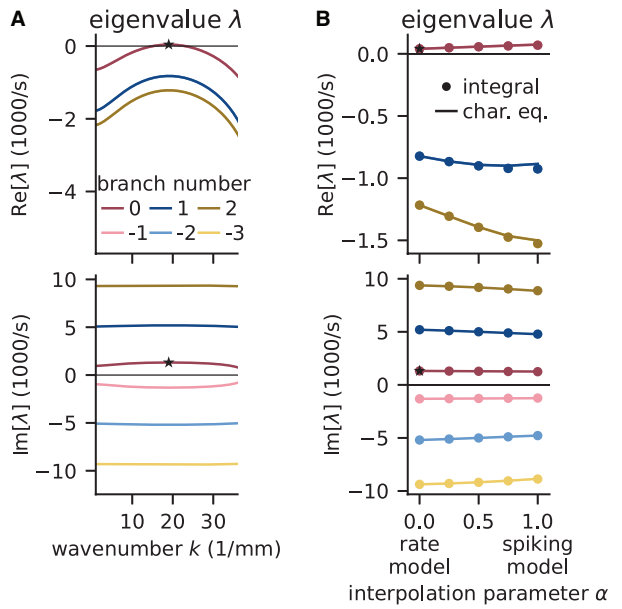

FIGURE 9 | Linear stability analysis of spatially structured network model. **(A)** Analytically exact solution for real (top) and imaginary (bottom) part of eigenvalue λ vs. wavenumber k using rate model derived by fit of spiking model at working point $(\mu, \sigma) = (10, 10)$ mV. Color-coded branches of Lambert W_B function; maximum real eigenvalue (star marker) on principal branch ($B = 0$). **(B)** Linear interpolation between rate ($\alpha = 0$) and spiking model ($\alpha = 1$) by numerical integration of Senk et al. (2020, Equation 30) (solid line) and by numerically solving the characteristic equation in Senk et al. (2020, Equation 29) (circular markers). Star markers at same data points as in **(A)**. Similar displays as in Senk et al. (2020, Figure 6).

argument of Senk et al. (2020), the predicted pattern formation could next be tested in a numerical simulation of the discrete spiking network model. Their Figure 7c for the delay $d = 1.5$ ms shows such results with the same parameters as used here; this figure also illustrates transitions from homogeneous states to oscillatory states by changing the delay (panels b, c, and e).

4. DISCUSSION

Mean-field theory grants important insights into the dynamics of neuronal networks. However, the lack of a publicly available numerical implementation for most methods entails a significant initial investment of time and effort prior to any scientific investigations. In this paper, we present the open-source toolbox NNMT, which currently focuses on methods for LIF neurons but is intended as a platform for collecting standard implementations of various neuronal network model analyses based on mean-field theory that have been thoroughly tested and validated by the neuroscientific community (Riquelme and Gjorgjieva, 2021). As use cases, we reproduce known results from the literature: the non-linear relation between the firing rates and the external input of an E-I-network (Sanzeni et al., 2020), the firing rates of a cortical microcircuit model, its response to oscillatory input, its power spectrum, and the identification of the connections that predominantly contribute to the model's low frequency oscillations (Schuecker et al., 2015; Bos et al., 2016), and pattern formation in a spiking network, analyzed by mapping it to a rate model and conducting a linear stability analysis (Senk et al., 2020).

In the remainder of the discussion, we compare NNMT to other tools suited for network model analysis. We expand on the different use cases of NNMT and also point out the inherent limitations of analytical methods for neuronal network analysis. We conclude with suggestions on how new tools can be added to NNMT and how the toolbox may grow and develop in the future.

4.1. Comparison to Other Tools

There are various approaches and corresponding tools that can help to gain a better understanding of a neuronal network model. There are numerous simulators that numerically solve the dynamical equations for concrete realizations of a network model and all its stochastic components, often focusing either on the resolution of single-neurons, for example NEST (Gewaltig and Diesmann, 2007), Brian (Stimberg et al., 2019), or Neuron (Hines and Carnevale, 2001), or on the population level, for example TheVirtualBrain (Sanz Leon et al., 2013). Similarly, general-purpose dynamical system software like XPPAUT (Ermentrout, 2002) can be used. Simulation tools, like DynaSim (Sherfey et al., 2018), come with enhanced functionality for simplifying batch analysis and parameter explorations. All these tools yield time-series of activity, and some of them even provide the methods for analyzing the generated data. However, simulations only indirectly link a model's parameters with its activity: to gain an understanding of how a model's parameters influence the statistics of their activity, it is necessary to run many simulations with different parameters and analyze the generated data subsequently.

Other approaches provide a more direct insight into a model's behavior on an abstract level: TheVirtualBrain and the Brain Dynamics Toolbox (Heitmann et al., 2018), for example, allow plotting a model's phase space vector field while the parameters can be changed interactively, allowing for exploration of low-dimensional systems defined by differential equations without the need for simulations. XPPAUT has an interface to AUTO-07P (Doedel and Oldeman, 1998), a software for performing numerical bifurcation and continuation analysis. It is worth noting that such tools are limited to models that are defined in terms of differential equations. Models specified in terms of update rules, such as binary neurons, need to be analyzed differently, for example using mean-field theory.

A third approach is to simplify the model analytically and simulate the simplified version. The simulation platform DiPDE[4] utilizes the population density approach to simulate the statistical evolution of a network model's dynamics. Schwalger et al. (2017) start from a microscopic model of generalized integrate-and-fire neurons and derive mesoscopic mean-field population equations, which reproduce the statistical and qualitative behavior of the homogeneous neuronal sub-populations. Similarly, Montbrió et al. (2015) derive a set of non-linear

[4]http://alleninstitute.github.io/dipde

differential equations describing the dynamics of the rate and mean membrane potentials of a population of quadratic integrate-and-fire (QIF) neurons. The simulation platform PyRates (Gast et al., 2019) provides an implementation of this QIF mean-field model, and allows simulating them to obtain the temporal evolution of the population activity measures.

However, mean-field and related theories can go beyond such reduced dynamical equations: they can directly link model parameters to activity statistics, and they can even provide access to informative network properties that might not be accessible otherwise. The spectral bound (Rajan and Abbott, 2006) of the effective connectivity matrix in linear response theory (Lindner et al., 2005; Pernice et al., 2011; Trousdale et al., 2012) is an example of such a property. It is a measure for the stability of the linearized system and determines, for example, the occurrence of slow dynamics and long-range correlations (Dahmen et al., 2022). Another example is the sensitivity measure presented in Section 3.3.3, which directly links a network model's connectivity with the properties of its power spectrum. These measures are not accessible via simulations. They require analytical calculation.

Similarly, NNMT is not a simulator. NNMT is a collection of mean-field equation implementations that directly relate a model's parameters to the statistics of its dynamics or to other informative properties. It provides these implementations in a format that makes them applicable to as many network models as possible. This is not to say that NNMT does not involve numerical integration procedures; solving self-consistent equations, such as in the case of the firing rates calculations in Section 3.2.1 and Section 3.2.2, is a common task, and a collection of respective solvers is part of NNMT.

4.2. Use Cases

In Section 3, we present concrete examples of how to apply some of the tools available. Here, we revisit some of the examples to highlight the use cases NNMT lends itself to, as well as provide some ideas for how the toolbox could be utilized in future projects.

Analytical methods have the advantage of being fast, and typically they only require a limited amount of computational resources. The computational costs for calculating analytical estimates of dynamical network properties like firing rates, as opposed to the costs of running simulations of a network model, are independent of the number of neurons the network is composed of. This is especially relevant for parameter space explorations, for which many simulations have to be performed. To speed up prototyping, a modeler can first perform a parameter scan using analytical tools from NNMT to get an estimate of the right parameter regimes and subsequently run simulations on this restricted set of parameters to arrive at the final model parameters. An example of such a parameter scan is given in Section 3.2.1, where the firing rates of a network are studied as a function of the external input.

Additionally to speeding up parameter space explorations, analytical methods may guide parameter space explorations in another way: namely, by providing an analytical relation between network model parameters and network dynamics, which allows a targeted adjustment of specific parameters to achieve a desired network activity. The prime example implemented in NNMT is the sensitivity measure presented in Section 3.3.3, which provides an intuitive relation between the network connectivity and the peaks of the power spectrum corresponding to the dominant oscillation frequencies. As shown in the final part of Section 3.3.3, the sensitivity measure identifies the connections which need to be adjusted in order to modify the dominant oscillation mode in a desired manner. This illustrates a mean-field method that provides a modeler with additional information about the origin of a model's dynamics, such that a parameter space exploration can be restricted to the few identified crucial model parameters.

A modeler investigating which features of a network model are crucial for the emergence of certain activity characteristics observed in simulations might be interested in comparing models of differing complexity. The respective mappings can be derived in mean-field theory, and one variant included in NNMT, which is presented in Section 3.4, allows mapping a LIF network to a simpler rate network. This is useful to investigate whether spiking dynamics is crucial for the observed phenomenon.

On a general note, which kind of questions researchers pursue is limited by and therefore depends on the tools they have at hand (Dyson, 2012). The availability of sophisticated neural network simulators for example has lead to the development of conceptually new and more complex neural network models, precisely because their users could focus on actual research questions instead of implementations. We hope that collecting useful implementations of analytical tools for network model analysis will have a similar effect on the development of new tools and that it might lead to new, creative ways of applying them.

4.3. Limitations

As a collection of analytical methods, NNMT comes with inherent limitations that apply to any toolbox for analytical methods: it is restricted to network, neuron, and synapse models, as well as observables, for which a mean-field theory exists, and the tools are based on analytical assumptions, simplifications, and approximations, restricting their valid parameter regimes and their explanatory power, which we expand upon in the next paragraphs.

Analytical methods can provide good estimates of network model properties, but there are limitations that must be considered when interpreting results provided by NNMT: First of all, the employed numerical solvers introduce numerical inaccuracies, but they can be remedied by changing hyperparameters such as integration step sizes or iteration termination thresholds. More importantly, analytical methods almost always rely on approximations, which can only be justified if certain assumptions are fulfilled. Typical examples of such assumptions are fast or slow synapses, or a random connectivity. If such assumptions are not met, at least approximately, and the valid parameter regime of a tool is left, the corresponding method is not guaranteed to give reliable results. Hence, it is important to be aware of a tool's limitations, which we aim to document as thoroughly as possible.

An important assumption of mean-field theory is uncorrelated Poissonian inputs. As discussed in Section 3.2.1, asynchronous irregular activity is a robust feature of inhibition

dominated networks, and mean-field theory is well-suited to describe the activity of such models. However, if a network model features highly correlated activity, or strong external input common to many neurons, approximating the input by uncorrelated noise no longer holds and mean-field estimates become unreliable.

In addition to the breakdown of such assumptions, some approaches, like linear response theory, rely on neglecting higher order terms. This restricts the tools' explanatory power, as they cannot predict higher order effects, such as the presence of higher harmonics in a network's power spectrum. Addressing these deficiencies necessitates using more elaborate analyses, and users should be aware of such limitations when interpreting the results.

Finally, a specific limitation of NNMT is that it currently only collects methods for LIF neurons. However, one of the aims of this paper is to encourage other scientists to contribute to the collection, and we outline how to do so in the following section.

4.4. How to Contribute and Outlook

A toolbox like NNMT always is an ongoing project, and there are various aspects that can be improved. In this section, we briefly discuss how available methods could be improved, what and how new tools could be added, as well as the benefits of implementing a new method with the help of NNMT.

First of all, NNMT in its current state is partly vectorized but the included methods are not parallelized, e.g., using multiprocessing or MPI for Python (`mpi4py`). Vectorization relies on `NumPy` (Harris et al., 2020) and `SciPy` (Virtanen et al., 2020), which are thread-parallel for specific backends, e.g., IntelMKL. With the tools available in the toolbox at the moment, run-time only becomes an issue in extensive parameter scans, for instance, when the transfer function needs to be calculated for a large range of frequencies. To further reduce the runtime, the code could be made fully vectorized. Alternatively, parallelization of many tools in NNMT is straightforward, as many of them include `for` loops over the different populations of a network model and `for` loops over the different analysis frequencies. A third option is just-in-time compilation, as provided by `Numba` (Lam et al., 2015).

Another aspect to consider is the range of network models a tool can be applied to. Thus far, the toolbox primarily supports arbitrary block structured networks. Future developments could extend the class of networks to even more general models.

Due to the research focus at our lab, NNMT presently mainly contains tools for LIF neurons in the fast synaptic regime and networks with random connectivity. Nonetheless, the structure of NNMT allows for adding methods for different neuron types, like for example binary (Ginzburg and Sompolinsky, 1994) or conductance-based neurons (Izhikevich, 2007; Richardson, 2007), as well as more elaborate network models. Another way to improve the toolbox is adding tools that complement the existing ones: As discussed in Section 4.3, many mean-field methods only give valid results for certain parameter ranges. Sometimes, there exist different approximations for the same quantity, valid in complementary parameter regimes. A concrete example is the currently implemented version of the transfer function for leaky integrate-and-fire neurons, based on Schuecker et al. (2015), which gives a good estimate for small synaptic time constants compared to the membrane time constant $\tau_s/\tau_m \ll 1$. A complementary estimate for $\tau_s/\tau_m \gg 1$ has been developed by Moreno-Bote and Parga (2006). Similarly, the current implementation of the firing rates of leaky integrate-and-fire neurons, based on the work of Fourcaud and Brunel (2002), is valid for $\tau_s/\tau_m \ll 1$. Recently, van Vreeswijk and Farkhooi (2019) have developed a method accurate for all combinations of synaptic and membrane time constants.

In the following, we explain how such implementations can be added and how using NNMT helps implementing new methods. Clearly, the implementations of NNMT help implementing methods that build on already existing ones. An example is the firing rate for LIF neurons with exponential synapses `nnmt.lif.exp._firing_rates()` which wraps the calculation of firing rates for LIF neurons with delta synapses `nnmt.lif.delta._firing_rates()`. Additionally, the toolbox may support the implementation of tools for other neuron models. As an illustration, let us consider adding the computation of the mean activity for a network of binary neurons (included in NNMT 1.1.0). We start with the equations for the mean input μ_a, its variance σ_a^2, and the firing rates $\boldsymbol{m}$ (Helias et al., 2014, Equations 4, 6, and 7)

$$\begin{aligned} \mu_a(\boldsymbol{m}) &= \sum_b K_{ab} J_{ab} m_b \,, \\ \sigma_a^2(\boldsymbol{m}) &= \sum_b K_{ab} J_{ab}^2 m_b (1 - m_b) \,, \\ m_a(\mu_a, \sigma_a) &= \frac{1}{2} \mathrm{erfc}\left(\frac{\Theta_a - \mu_a}{\sqrt{2}\sigma_a} \right) , \end{aligned} \tag{19}$$

with indegree matrix K_{ab} from population b to population a, synaptic weight matrix J_{ab}, and firing-threshold Θ_a. The sum $\sum_b$ may include an external population providing input to the model. This set of self-consistent equations has the same structure as the self-consistent equations for the firing rates of a network of LIF neurons, Equation (8): the input statistics are given as functions of the rate, and the rate is given as a function of the input statistics. Therefore, it is possible to reuse the firing rate integration procedure for LIF neurons, providing immediate access to the two different methods presented in Section 3.2.1. Accordingly, it is sufficient to implement Equation (19) in a new submodule `nnmt.binary` and apply the solver provided by NNMT to extend the toolbox to binary neurons.

The above example demonstrates the benefits of collecting analytical tools for network model analysis in a common framework. The more methods and corresponding solvers the toolbox comprises, the easier implementing new methods becomes. Therefore, contributions to the toolbox are highly welcome; this can be done via the standard pull request workflow on GitHub (see the "Contributors guide" of the official documentation of NNMT[2]). We hope that in the future, many scientists will contribute to this collection of analytical methods for neuronal network model analysis, such that, at some point, we will have tools from all parts of mean-field theory

of neuronal networks, made accessible in a usable format to all neuroscientists.

DATA AVAILABILITY STATEMENT

Publicly available datasets were used in this study, and the corresponding sources are cited in the main text. The toolbox's repository can be found at https://github.com/INM-6/nnmt, and the parameter files used in the presented examples can be found in the examples section of the online documentation https://nnmt.readthedocs.io/en/latest/.

AUTHOR CONTRIBUTIONS

HB and MH developed and implemented the code base and the initial version of the toolbox. ML, JS, and SE designed the current version of the toolbox. ML implemented the current version of the toolbox, vectorized and generalized tools, developed and implemented the test suite, wrote the documentation, and created the example shown in Section 3.2.2. AM improved the numerics of the firing rate integration (Methods) and created the example shown in Section 3.2.1. SE implemented integration tests, improved the functions related to the `sensitivity_measure`, and created the examples shown in Section 3.3. JS developed and implemented the tools used in Section 3.4 and created the respective example. ML, JS, SE, AM, and MH wrote this article. All authors approved the submitted version.

FUNDING

This project has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under Specific Grant Agreement Nos. 720270 (HBP SGA1), 785907 (HBP SGA2), and 945539 (HBP SGA3), has been partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 368482240/GRK2416, and has been partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 491111487. This research was supported by the Joint Lab "Supercomputing and Modeling for the Human Brain".

ACKNOWLEDGMENTS

We would like to thank Jannis Schuecker, who has contributed to the development and implementation of the code base and the initial version of the toolbox, and Angela Fischer, who supported us designing **Figure 1**. Additionally, we would also like to thank our reviewers for the thorough and constructive feedback, which lead to significant improvements.

REFERENCES

Abramowitz, M., and Stegun, I. A. (1974). *Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables* (New York: Dover Publications).

Ahmadian, Y., and Miller, K. D. (2021). What is the dynamical regime of cerebral cortex? *Neuron* 109, 3373–3391. doi: 10.1016/j.neuron.2021.07.031

Amari, S.-I. (1975). Homogeneous nets of neuron-like elements. *Biol. Cybern.* 17, 211–220. doi: 10.1007/BF00339367

Amari, S.-I. (1977). Dynamics of pattern formation in lateral-inhibition type neural fields. *Biol. Cybern.* 27, 77–87. doi: 10.1007/bf00337259

Amit, D. J., and Brunel, N. (1997a). Dynamics of a recurrent network of spiking neurons before and following learning. *Netw. Comp. Neural Sys.* 8, 373–404. doi: 10.1088/0954-898x_8_4_003

Amit, D. J. and Brunel, N. (1997b). Model of global spontaneous activity and local structured activity during delay periods in the cerebral cortex. *Cereb. Cortex* 7, 237–252. doi: 10.1093/cercor/7.3.237

Amit, D. J., and Tsodyks, M. V. (1991). Quantitative study of attractor neural network retrieving at low spike rates I: substrate–spikes, rates and neuronal gain. *Network* 2, 259. doi: 10.1088/0954-898X_2_3_003

Bos, H., Diesmann, M., and Helias, M. (2016). Identifying anatomical origins of coexisting oscillations in the cortical microcircuit. *PLOS Comput. Biol.* 12, e1005132. doi: 10.1371/journal.pcbi.1005132

Braitenberg, V. and Schüz, A. (1998). *Cortex: Statistics and Geometry of Neuronal Connectivity, 2nd Edn.* Berlin: Springer-Verlag.

Bressloff, P. C. (2012). Spatiotemporal dynamics of continuum neural fields. *J. Phys. A* 45, 033001. doi: 10.1088/1751-8113/45/3/033001

Bressloff, P. C., Cowan, J. D., Golubitsky, M., Thomas, P. J., and Wiener, M. C. (2001). Geometric visual hallucinations, euclidean symmetry and the functional architecture of striate cortex. *Phil. Trans. R. Soc. B* 356, 299–330. doi: 10.1098/rstb.2000.0769

Brunel, N. (2000). Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. *J. Comput. Neurosci.* 8, 183–208. doi: 10.1023/a:1008925309027

Brunel, N., Chance, F. S., Fourcaud, N., and Abbott, L. F. (2001). Effects of synaptic noise and filtering on the frequency response of spiking neurons. *Phys. Rev. Lett.* 86, 2186–2189. doi: 10.1103/physrevlett.86.2186

Brunel, N., and Hakim, V. (1999). Fast global oscillations in networks of integrate-and-fire neurons with low firing rates. *Neural Comput.* 11, 1621–1671. doi: 10.1162/089976699300016179

Brunel, N., and Latham, P. (2003). Firing rate of the noisy quadratic integrate-and-fire neuron. *Neural Comput.* 15, 2281–2306. doi: 10.1162/089976603322362365

Buice, M. A., and Chow, C. C. (2013). Beyond mean field theory: statistical field theory for neural networks. *J. Stat. Mech.* 2013, P03003. doi: 10.1088/1742-5468/2013/03/P03003

Coombes, S. (2005). Waves, bumps, and patterns in neural field theories. *Biol. Cybern.* 93, 91–108. doi: 10.1007/s00422-005-0574-y

Coombes, S., bei Graben, P., Potthast, R., and Wright, J. (2014). *Neural Fields. Theory and Applications.* Berlin; Heidelberg: Springer-Verlag.

Corless, R. M., Gonnet, G. H., Hare, D. E. G., Jeffrey, D. J., and Knuth, D. E. (1996). On the lambert w function. *Adv. Comput. Math.* 5, 329–359. doi: 10.1007/BF02124750

Dahmen, D., Layer, M., Deutz, L., Dąbrowska, P. A., Voges, N., von Papen, M., et al. (2022). Global organization of neuronal activity only requires unstructured local connectivity. *eLife* 11, e68422. doi: 10.7554/eLife.68422.sa0

Dasbach, S., Tetzlaff, T., Diesmann, M., and Senk, J. (2021). Dynamical characteristics of recurrent neuronal networks are robust against low synaptic weight resolution. *Front. Neurosci.* 15, 757790. doi: 10.3389/fnins.2021.757790

DeFelipe, J., Alonso-Nanclares, L., and Arellano, J. (2002). Microstructure of the neocortex: comparative aspects. *J. Neurocytol.* 31, 299–316. doi: 10.1023/A:1024130211265

Doedel, E. J., and Oldeman, B. (1998). *Auto-07p: Continuation and Bifurcation Software.* Montreal, QC: Concordia University Canada

Dyson, F. J. (2012). Is science mostly driven by ideas or by tools? *Science* 338, 1426–1427. doi: 10.1126/science.1232773

Ermentrout, B. (2002). *Simulating, Analyzing, and Animating Dynamical Systems: A Guide to Xppaut for Researchers and Students (Software, Environments, Tools)*. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Ermentrout, G. B., and Cowan, J. D. (1979). A mathematical theory of visual hallucination patterns. *Biol. Cybern.* 34, 137–150. doi: 10.1007/BF00336965

Fourcaud, N., and Brunel, N. (2002). Dynamics of the firing probability of noisy integrate-and-fire neurons. *Neural Comput.* 14, 2057–2110. doi: 10.1162/089976602320264015

Fourcaud-Trocmé, N., Hansel, D., van Vreeswijk, C., and Brunel, N. (2003). How spike generation mechanisms determine the neuronal response to fluctuating inputs. *J. Neurosci.* 23, 11628–11640. doi: 10.1523/JNEUROSCI.23-37-11628.2003

Gast, R., Rose, D., Salomon, C., Möller, H. E., Weiskopf, N., and Knösche, T. R. (2019). Pyrates - a python framework for rate-based neural simulations. *PLoS ONE* 14, e0225900. doi: 10.1371/journal.pone.0225900

Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014). *Neuronal Dynamics. From Single Neurons to Networks and Models of Cognition*. Cambridge: Cambridge University Press.

Gewaltig, M.-O., and Diesmann, M. (2007). NEST (nEural simulation tool). *Scholarpedia* 2, 1430. doi: 10.4249/scholarpedia.1430

Giese, M. A. (2012). *Dynamic Neural Field Theory for Motion Perception, Vol. 469*. Berlin; Heidelberg: Springer Science & Business Media)

Ginzburg, I., and Sompolinsky, H. (1994). Theory of correlations in stochastic neural networks. *Phys. Rev. E* 50, 3171–3191. doi: 10.1103/PhysRevE.50.3171

Goldenfeld, N. (1992). *Lectures on Phase Transitions and the Renormalization Group*. Reading, MA: Perseus books.

Golosio, B., Tiddia, G., Luca, C. D., Pastorelli, E., Simula, F., and Paolucci, P. S. (2021). Fast simulations of highly-connected spiking cortical models using GPUs. *Front. Comput. Neurosci.* 15, 627620. doi: 10.3389/fncom.2021.627620

Grabska-Barwinska, A., and Latham, P. (2014). How well do mean field theories of spiking quadratic-integrate-and-fire networks work in realistic parameter regimes? *J. Comput. Neurosci.* 36, 469–481. doi: 10.1007/s10827-013-0481-5

Grytskyy, D., Tetzlaff, T., Diesmann, M., and Helias, M. (2013). A unified view on weakly correlated recurrent networks. *Front. Comput. Neurosci.* 7, 131. doi: 10.3389/fncom.2013.00131

Hagen, E., Dahmen, D., Stavrinou, M. L., Lindén, H., Tetzlaff, T., van Albada, S. J., et al. (2016). Hybrid scheme for modeling local field potentials from point-neuron networks. *Cereb. Cortex* 26, 4461–4496. doi: 10.1093/cercor/bhw237

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., et al. (2020). Array programming with NumPy. *Nature* 585, 357–362. doi: 10.1038/s41586-020-2649-2

Heitmann, S., Aburn, M. J., and Breakspear, M. (2018). The brain dynamics toolbox for matlab. *Neurocomputing* 315, 82–88. doi: 10.1016/j.neucom.2018.06.026

Helias, M., Tetzlaff, T., and Diesmann, M. (2014). The correlation structure of local cortical networks intrinsically results from recurrent dynamics. *PLoS Comput. Biol.* 10, e1003428. doi: 10.1371/journal.pcbi.1003428

Hertz, J. (2010). Cross-correlations in high-conductance states of a model cortical network. *Neural Comput.* 22, 427–447. doi: 10.1162/neco.2009.06-08-806

Hines, M. L., and Carnevale, N. T. (2001). NEURON: a tool for neuroscientists. *Neuroscientist* 7, 123–135. doi: 10.1177/107385840100700207

Izhikevich, E. M. (2007). *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*. Cambridge, MA: MIT Press.

Jirsa, V. K., and Haken, H. (1996). Field theory of electromagnetic brain activity. *Phys. Rev. Lett.* 77, 960. doi: 10.1103/PhysRevLett.77.960

Jirsa, V. K., and Haken, H. (1997). A derivation of a macroscopic field theory of the brain from the quasi-microscopic neural dynamics. *Phys. D* 99, 503–526. doi: 10.1016/S0167-2789(96)00166-2

Knight, J. C., and Nowotny, T. (2018). GPUs outperform current HPC and neuromorphic solutions in terms of speed and energy when simulating a highly-connected cortical model. *Front. Neurosci.* 12, 941. doi: 10.3389/fnins.2018.00941

Laing, C. R., and Troy, W. C. (2003). Two-bump solutions of amari-type models of neuronal pattern formation. *Phys. D* 178, 190–218. doi: 10.1016/S0167-2789(03)00013-7

Laing, C. R., Troy, W. C., Gutkin, B., and Ermentrout, B. G. (2002). Multiple bumps in a neuronal model of working memory. *SIAM J. Appl. Math.* 63, 62–97. doi: 10.1137/s0036139901389495

Lam, S. K., Pitrou, A., and Seibert, S. (2015). "Numba: a llvm-based python jit compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, Austin, TX, 1–6

Layer, M., Senk, J., Essink, S., van Meegen, A., Bos, H., and Helias, M. (2021). NNMT (1.0.0). *Zenodo*. doi: 10.5281/zenodo.5779548

Lindner, B. (2004). Interspike interval statistics of neurons driven by colored noise. *Phys. Rev. E* 69, 0229011–0229014. doi: 10.1103/PhysRevE.69.022901

Lindner, B., Doiron, B., and Longtin, A. (2005). Theory of oscillatory firing induced by spatially correlated noise and delayed inhibitory feedback. *Phys. Rev. E* 72, 061919. doi: 10.1103/physreve.72.061919

Lindner, B., and Longtin, A. (2005). Effect of an exponentially decaying threshold on the firing statistis of a stochastic integate-and-fire neuron. *J. Theor. Biol.* 232, 505–521. doi: 10.1016/j.jtbi.2004.08.030

Lindner, B., and Schimansky-Geier, L. (2001). Transmission of noise coded versus additive signals through a neuronal ensemble. *Phys. Rev. Lett.* 86, 2934–2937. doi: 10.1103/physrevlett.86.2934

Mattia, M., Biggio, M., Galluzzi, A., and Storace, M. (2019). Dimensional reduction in networks of non-markovian spiking neurons: Equivalence of synaptic filtering and heterogeneous propagation delays. *PLoS Comput. Biol.* 15, e1007404. doi: 10.1371/journal.pcbi.1007404

Montbrió, E., Pazó, D., and Roxin, A. (2015). Macroscopic description for networks of spiking neurons. *Phys Rev X* 5, 021028. doi: 10.1103/PhysRevX.5.021028

Moreno-Bote, R., and Parga, N. (2006). Auto- and crosscorrelograms for the spike response of leaky integrate-and-fire neurons with slow synapses. *Phys. Rev. Lett.* 96, 028101. doi: 10.1103/PhysRevLett.96.028101

Nunez, P. L. (1974). The brain wave equation: a model for the eeg. *Math. Biosci.* 21, 279–297. doi: 10.1016/0025-5564(74)90020-0

Olver, F. W. J., Olde Daalhuis, A. B., Lozier, D. W., Schneider, B. I., Boisvert, R. F., Clark, C. W., et al. (2021). *NIST Digital Library of Mathematical Functions*. Available online at: http://dlmf.nist.gov/

Ostojic, S. (2014). Two types of asynchronous activity in networks of excitatory and inhibitory spiking neurons. *Nat. Neurosci.* 17, 594–600. doi: 10.1038/nn.3658

Ostojic, S., and Brunel, N. (2011). From spiking neuron models to linear-nonlinear models. *PLoS Comput. Biol.* 7, e1001056. doi: 10.1371/journal.pcbi.1001056

Pernice, V., Staude, B., Cardanobile, S., and Rotter, S. (2011). How structure determines correlations in neuronal networks. *PLoS Comput. Biol.* 7, e1002059. doi: 10.1371/journal.pcbi.1002059

Potjans, T. C., and Diesmann, M. (2014). The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model. *Cereb. Cortex* 24, 785–806. doi: 10.1093/cercor/bhs358

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2007). *Numerical Recipes: The Art of Scientific Computing*, 3rd edn. Cambridge University Press.

Rajan, K., and Abbott, L. F. (2006). Eigenvalue spectra of random matrices for neural networks. *Phys. Rev. Lett.* 97, 188104. doi: 10.1103/PhysRevLett.97.188104

Renart, A., De La Rocha, J., Bartho, P., Hollender, L., Parga, N., Reyes, A., et al. (2010). The asynchronous state in cortical circuits. *Science* 327, 587–590. doi: 10.1126/science.1179850

Richardson, M. J. E. (2007). Firing-rate response of linear and nonlinear integrate-and-fire neurons to modulated current-based and conductance-based synaptic drive. *Phys. Rev. E* 76, 1–15. doi: 10.1103/PhysRevE.76.021919

Richardson, M. J. E. (2008). Spike-train spectra and network response functions for non-linear integrate-and-fire neurons. *Biol. Cybern.* 99, 381–392. doi: 10.1007/s00422-008-0244-y

Riquelme, J. L., and Gjorgjieva, J. (2021). Towards readable code in neuroscience. *Nat. Rev. Neurosci.* 22, 257–258. doi: 10.1038/s41583-021-00450-y

Rosenbaum, R. and Doiron, B. (2014). Balanced networks of spiking neurons with spatially dependent recurrent connections. *Phys. Rev. X* 4, 021039. doi: 10.1103/PhysRevX.4.021039

Rosenbaum, R., Smith, M. A., Kohn, A., Rubin, J. E., and Doiron, B. (2017). The spatial structure of correlated neuronal variability. *Nat. Neurosci.* 20, 107–114. doi: 10.1038/nn.4433

Sanz Leon, P., Knock, S., Woodman, M., Domide, L., Mersmann, J., McIntosh, A., et al. (2013). The virtual brain: a simulator of primate brain network dynamics. *Front. Neuroinform.* 7, 10. doi: 10.3389/fninf.2013.00010

Sanzeni, A., Histed, M. H., and Brunel, N. (2020). Response nonlinearities in networks of spiking neurons. *PLOS Comput. Biol.* 16, e1008165. doi: 10.1371/journal.pcbi.1008165

Schmidt, M., Bakker, R., Hilgetag, C. C., Diesmann, M., and van Albada, S. J. (2018). Multi-scale account of the network structure of macaque visual cortex. *Brain Struct. Func.* 223, 1409–1435. doi: 10.1007/s00429-017-1554-4

Schöner, G. (2008). "Dynamical systems approaches to cognition," in *Cambridge Handbook of Computational Cognitive Modeling*. Cambridge: Cambridge University Press, 101–126.

Schuecker, J., Diesmann, M., and Helias, M. (2014). Reduction of colored noise in excitable systems to white noise and dynamic boundary conditions. *arXiv[Preprint].arXiv:1410.8799*. doi: 10.48550/arXiv.1410.8799

Schuecker, J., Diesmann, M., and Helias, M. (2015). Modulated escape from a metastable state driven by colored noise. *Phys. Rev. E* 92, 052119. doi: 10.1103/PhysRevE.92.052119

Schuecker, J., Goedeke, S., and Helias, M. (2018). Optimal sequence memory in driven random networks. *Phys. Rev. X* 8, 041029. doi: 10.1103/PhysRevX.8.041029

Schwalger, T., Deger, M., and Gerstner, W. (2017). Towards a theory of cortical columns: From spiking neurons to interacting neural populations of finite size. *PLoS Comput. Biol.* 13, e1005507. doi: 10.1371/journal.pcbi.1005507

Schwalger, T., Droste, F., and Lindner, B. (2015). Statistical structure of neural spiking under non-poissonian or other non-white stimulation. *J. Comput. Neurosci.* 39, 29. doi: 10.1007/s10827-015-0560-x

Sejnowski, T. (1976). On the stochastic dynamics of neuronal interaction. *Biol. Cybern.* 22, 203–211. doi: 10.1007/BF00365086

Senk, J., Korvasová, K., Schuecker, J., Hagen, E., Tetzlaff, T., Diesmann, M., et al. (2020). Conditions for wave trains in spiking neural networks. *Phys. Rev. Res.* 2, 023174. doi: 10.1103/physrevresearch.2.023174

Senk, J., Kriener, B., Djurfeldt, M., Voges, N., Jiang, H.-J., Schüttler, L., et al. (in press). Connectivity concepts in neuronal network modeling. *PLOS Comput. Biol*.

Sherfey, J. S., Soplata, A. E., Ardid, S., Roberts, E. A., Stanley, D. A., Pittman-Polletta, B. R., et al. (2018). Dynasim: a matlab toolbox for neural modeling and simulation. *Front. Neuroinform.* 12, 10. doi: 10.3389/fninf.2018.00010

Siegert, A. J. (1951). On the first passage time probability problem. *Phys. Rev.* 81, 617–623. doi: 10.1103/PhysRev.81.617

Sompolinsky, H., Crisanti, A., and Sommers, H. J. (1988). Chaos in random neural networks. *Phys. Rev. Lett.* 61, 259–262. doi: 10.1103/PhysRevLett.61.259

Stiller, J., and Radons, G. (1998). Dynamics of nonlinear oscillators with random interactions. *Phys. Rev. E* 58, 1789. doi: 10.1103/PhysRevE.58.1789

Stimberg, M., Brette, R., and Goodman, D. F. (2019). Brian 2, an intuitive and efficient neural simulator. *eLife* 8, e47314. doi: 10.7554/elife.47314

Tetzlaff, T., Helias, M., Einevoll, G. T., and Diesmann, M. (2012). Decorrelation of neural-network activity by inhibitory feedback. *PLOS Comput. Biol.* 8, e1002596. doi: 10.1371/journal.pcbi.1002596

Toyoizumi, T., and Abbott, L. F. (2011). Beyond the edge of chaos: Amplification and temporal integration by recurrent networks in the chaotic regime. *Phys. Rev. E* 84, 051908. doi: 10.1103/PhysRevE.84.051908

Trousdale, J., Hu, Y., Shea-Brown, E., and Josic, K. (2012). Impact of network structure and cellular response on spike time correlations. *PLoS Comput. Biol.* 8, e1002408. doi: 10.1371/journal.pcbi.1002408

Tuckwell, H. C. (1988). *Introduction to Theoretical Neurobiology, Vol. 2* Cambridge: Cambridge University Press.

van Albada, S. J., Rowley, A. G., Senk, J., Hopkins, M., Schmidt, M., Stokes, A. B., et al. (2018). Performance comparison of the digital neuromorphic hardware SpiNNaker and the neural network simulation software NEST for a full-scale cortical microcircuit model. *Front. Neurosci.* 12, 291. doi: 10.3389/fnins.2018.00291

van Meegen, A., and Lindner, B. (2018). Self-consistent correlations of randomly coupled rotators in the asynchronous state. *Phys. Rev. Lett.* 121, 258302. doi: 10.1103/PhysRevLett.121.258302

van Vreeswijk, C., and Farkhooi, F. (2019). Fredholm theory for the mean first-passage time of integrate-and-fire oscillators with colored noise input. *Phys. Rev. E* 100, 060402. doi: 10.1103/PhysRevE.100.060402

van Vreeswijk, C., and Sompolinsky, H. (1996). Chaos in neuronal networks with balanced excitatory and inhibitory activity. *Science* 274, 1724–1726. doi: 10.1126/science.274.5293.1724

van Vreeswijk, C., and Sompolinsky, H. (1998). Chaotic balanced state in a model of cortical circuits. *Neural Comput.* 10, 1321–1371. doi: 10.1162/089976698300017214

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., et al. (2020). SciPy 1.0: fundamental algorithms for scientific computing in python. *Nat. Methods* 17, 261–272. doi: 10.1038/s41592-019-0686-2

Wagatsuma, N., Potjans, T. C., Diesmann, M., and Fukai, T. (2011). Layer-dependent attentional processing by top-down signals in a visual cortical microcircuit model. *Front. Comput. Neurosci.* 5, 31. doi: 10.3389/fncom.2011.00031

Wilson, H. R., and Cowan, J. D. (1972). Excitatory and inhibitory interactions in localized populations of model neurons. *Biophys. J.* 12, 1 – 24. doi: 10.1016/S0006-3495(72)86068-5

Wilson, H. R., and Cowan, J. D. (1973). A mathematical theory of the functional dynamics of cortical and thalamic nervous tissue. *Kybernetik* 13, 55–80. doi: 10.1007/BF00288786

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Copyright © 2022 Layer, Senk, Essink, van Meegen, Bos and Helias. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

A. APPENDIX

A.1. Siegert Implementation

Here, we describe how we solve the integral in Equation (4) numerically in a fully vectorized manner. The difficulty in Equation (4), $\phi(\mu,\sigma) = 1/[\tau_r + \tau_m\sqrt{\pi} I(\widetilde{V}_0, \widetilde{V}_{th})]$ where $\widetilde{V}_0 = \widetilde{V}_0(\mu,\sigma)$ and $\widetilde{V}_{th} = \widetilde{V}_{th}(\mu,\sigma)$ are determined by either Equation (5) or Equation (10), is posed by the integral

$$I(\widetilde{V}_0, \widetilde{V}_{th}) = \int_{\widetilde{V}_0}^{\widetilde{V}_{th}} e^{s^2}(1 + \mathrm{erf}(s))\mathrm{d}s. \tag{A1}$$

This integral is problematic due to the multiplication of e^{s^2} and $1 + \mathrm{erf}(s)$ in the integrand which leads to overflow and loss of significance.

To address this, we split the integral into different domains depending on the sign of the integration variable. Furthermore, we use the scaled complementary error function

$$\mathrm{erf}(s) = 1 - e^{-s^2}\mathrm{erfcx}(s) \tag{A2}$$

to extract the leading exponential contribution. Importantly, $\mathrm{erfcx}(s)$ decreases monotonically from $\mathrm{erfcx}(0) = 1$ with power law asymptotics $\mathrm{erfcx}(s) \sim 1/(\sqrt{\pi}s)$, hence it does not contain any exponential contribution. For positive s, the exponential contribution in the prefactor of $\mathrm{erfcx}(s)$ cancels the e^{s^2} factor in the integrand. For negative s, the integrand simplifies even further to $e^{s^2}(1 + \mathrm{erf}(-s)) = \mathrm{erfcx}(s)$ using $\mathrm{erf}(-s) = -\mathrm{erf}(s)$. In addition to $\mathrm{erfcx}(s)$, we employ the Dawson function

$$D(s) = e^{-s^2}\int_0^s e^{r^2}\mathrm{d}r \tag{A3}$$

to solve some of the integrals analytically. The Dawson function has a power law tail, $D(s) \sim 1/(2s)$; hence, it also does not carry an exponential contribution. Both $\mathrm{erfcx}(s)$ and the Dawson function are fully vectorized in `SciPy` (Virtanen et al., 2020).

Any remaining integrals are solved using Gauss–Legendre quadrature (Press et al., 2007). By construction, Gauss–Legendre quadrature of order k solves integrals of polynomials up to degree k on the interval $[-1, 1]$ exactly. Thus, it gives very good results if the integrand is well approximated by a polynomial of degree k. The quadrature rule itself is

$$\int_a^b f(s)\mathrm{d}s \approx \frac{b-a}{2}\sum_{i=1}^{k} w_i f\left(\frac{b-a}{2}u_i + \frac{b+a}{2}\right), \tag{A4}$$

where the u_i are the roots of the Legendre polynomial of order k and the w_i are appropriate weights such that a polynomial of degree k is integrated exactly. We use a fixed order quadrature for which Equation (A4) is straightforward to vectorize to multiple a and b. We determine the order of the quadrature iteratively by comparison with an adaptive quadrature rule; usually, a small order $k = O(10)$ already yields very good results for an $\mathrm{erfcx}(s)$ integrand.

Inhibitory Regime

First, we consider the case where lower and upper bound of the integral are positive, $0 < \widetilde{V}_0 < \widetilde{V}_{th}$. This corresponds to strongly inhibitory mean input. Expressing the integrand in terms of $\mathrm{erfcx}(s)$ and using the Dawson function, we get

$$I_{inh}(\widetilde{V}_0, \widetilde{V}_{th}) = 2e^{\widetilde{V}_{th}^2}D(\widetilde{V}_{th}) - 2e^{\widetilde{V}_0^2}D(\widetilde{V}_0) - \int_{\widetilde{V}_0}^{\widetilde{V}_{th}}\mathrm{erfcx}(s)\mathrm{d}s.$$

The remaining integral is evaluated using Gauss–Legendre quadrature, Equation (A4). We extract the leading contribution $e^{\widetilde{V}_{th}^2}$ from the denominator in Equation (4) and arrive at

$$\phi(\mu,\sigma) = \frac{e^{-\widetilde{V}_{th}^2}}{\tau_r e^{-\widetilde{V}_{th}^2} + \tau_m\sqrt{\pi}\left(e^{-\widetilde{V}_{th}^2}I_{inh}(\widetilde{V}_0, \widetilde{V}_{th})\right)}. \tag{A5}$$

Extracting $e^{\widetilde{V}_{th}^2}$ from the denominator reduces the latter to $2\tau_m\sqrt{\pi}D(\widetilde{V}_{th})$ and exponentially small correction terms (remember $0 < \widetilde{V}_0 < \widetilde{V}_{th}$ because $V_0 < V_{th}$), thereby preventing overflow.

Excitatory Regime

Second, we consider the case where lower and upper bound of the integral are negative, $\widetilde{V}_0 < \widetilde{V}_{th} < 0$. This corresponds to strongly excitatory mean input. In this regime, we change variables $s \to -s$ to make the domain of integration positive. Using $\mathrm{erf}(-s) = -\mathrm{erf}(s)$ as well as $\mathrm{erfcx}(s)$, we get

$$I_{exc}(\widetilde{V}_0, \widetilde{V}_{th}) = \int_{|\widetilde{V}_{th}|}^{|\widetilde{V}_0|}\mathrm{erfcx}(s)\mathrm{d}s.$$

Thus, we evaluate Equation (4) as

$$\phi(\mu,\sigma) = \frac{1}{\tau_r + \tau_m\sqrt{\pi}\int_{|\widetilde{V}_{th}|}^{|\widetilde{V}_0|}\mathrm{erfcx}(s)\mathrm{d}s}. \tag{A6}$$

In particular, there is no exponential contribution involved in this regime.

Intermediate Regime

Last, we consider the remaining case $\widetilde{V}_0 \le 0 \le \widetilde{V}_{th}$. We split the integral at zero and use the previous steps for the respective parts to get

$$I_{interm}(\widetilde{V}_0, \widetilde{V}_{th}) = 2e^{\widetilde{V}_{th}^2}D(\widetilde{V}_{th}) + \int_{\widetilde{V}_{th}}^{|\widetilde{V}_0|}\mathrm{erfcx}(s)\mathrm{d}s.$$

Note that the sign of the second integral depends on whether $|\widetilde{V}_0| > \widetilde{V}_{th}$ (+) or not (−). Again, we extract the leading contribution $e^{\widetilde{V}_{th}^2}$ from the denominator in Equation (4) and arrive at

$$\phi(\mu,\sigma) = \frac{e^{-\widetilde{V}_{th}^2}}{\tau_r e^{-\widetilde{V}_{th}^2} + \tau_m\sqrt{\pi}\left(e^{-\widetilde{V}_{th}^2}I_{interm}(\widetilde{V}_0, \widetilde{V}_{th})\right)}. \tag{A7}$$

As before, extracting $e^{\widetilde{V}_{th}^2}$ from the denominator prevents overflow.

Deterministic Limit

The deterministic limit $\sigma \to 0$ corresponds to $|\widetilde{V}_0|, |\widetilde{V}_{\mathrm{th}}| \to \infty$ for both Equation (5) and Equation (10). In the inhibitory and the intermediate regime, we see immediately that $\phi(\mu, \sigma \to 0) \to 0$ due to the dominant contribution $e^{-\widetilde{V}_{\mathrm{th}}^2}$. In the excitatory regime, we use the asymptotics $\mathrm{erfcx}(s) \sim 1/(\sqrt{\pi}s)$ to get

$$I(\widetilde{V}_0, \widetilde{V}_{\mathrm{th}}) \to \int_{|\widetilde{V}_{\mathrm{th}}|}^{|\widetilde{V}_0|} \frac{1}{\sqrt{\pi}s} \mathrm{d}s = \frac{1}{\sqrt{\pi}} \ln \frac{|\widetilde{V}_0|}{|\widetilde{V}_{\mathrm{th}}|}.$$

Inserting this into Equation (4) yields

$$\phi(\mu, \sigma) \to \begin{cases} \frac{1}{\tau_{\mathrm{r}} + \tau_{\mathrm{m}} \ln \frac{\mu - V_0}{\mu - V_{\mathrm{th}}}} & \text{if } \mu > V_{\mathrm{th}} \\ 0 & \text{otherwise} \end{cases}, \tag{A8}$$

which is the firing rate of a leaky integrate-and-fire neuron driven by a constant input (Gerstner et al., 2014). Thus, this implementation also tolerates the deterministic limit of a very small noise intensity σ.

TABLE A1 | Microcircuit Parameters.

Symbol	Value (Potjans and Diesmann, 2014)	Value (Bos et al., 2016)	Description
$K_{4E,4I}$	795	675	In-degree from 4I to 4E
$K_{4E,ext}$	2100	1780	External in-degree to 4E
$D(\omega)$	none	truncated Gaussian	Delay distribution
$d_e \pm \delta d_e$	1.5 ± 0.75 ms	1.5 ± 1.5 ms	Mean and standard deviation of excitatory delay
$d_i \pm \delta d_i$	0.75 ± 0.375 ms	0.75 ± 0.75 ms	Mean and standard deviation of inhibitory delay

Parameter adaptions used here are introduced by Bos et al. (2016) compared to original microcircuit model. K_{ij} denotes the in-degrees from population j to population i. The delays in the simulated networks were drawn from a truncated Gaussian distribution with the given mean and standard deviation. The mean-field approximation of the microcircuit by Potjans and Diesmann (2014) assumes the delay to be fixed at the mean value, which is specified in the toolbox by setting the parameter `delay_dist` *to* `none`.

A.2. Transfer Function Notations

In Section 3.3.1 we introduce the analytical form of the transfer function implemented in the toolbox. Schuecker et al. (2015), derive a more general form of the transfer function, which includes a modulation of the variance of the input. Here we compare the notation used in Equation (11) to the notation used in Schuecker et al. (2015, Eq. 29).

Schuecker et al. (2015) define the modulations of input mean and variance as

$$\begin{aligned} \mu(t) &= \mu + \epsilon\mu\, \mathrm{e}^{\mathrm{i}\omega t}, \\ \sigma^2(t) &= \sigma^2 + H\sigma^2\, \mathrm{e}^{\mathrm{i}\omega t}, \end{aligned} \tag{A9}$$

and introduce the transfer function in terms of its influence on the firing rate

$$\nu(t)/\nu_0 = 1 + n(\omega)\, \mathrm{e}^{\mathrm{i}\omega t},$$

where ν_0 is the stationary firing rate. Here the transfer function $n(\omega)$ includes contributions of both the modulation of the mean $n_G(\omega) \propto \epsilon$ and the modulation of the variance $n_H(\omega) \propto H$. We write the modulation of the mean as

$$\mu(t) = \mu + \delta\mu\, \mathrm{e}^{\mathrm{i}\omega t},$$

implying that $\delta\mu$ corresponds to $\epsilon\mu$ in Equation (A9). As we only consider the modulation of the mean, the firing rate can be rewritten as

$$\nu(t) = \nu + N(\omega)\, \delta\mu\, \mathrm{e}^{\mathrm{i}\omega t},$$

where we moved the stationary firing rate ν to the right hand side and included it in the definition of the transfer function $N(\omega)$. In the main text we emphasize that $\mu(t)$ and $\nu(t)$ are physical quantities by only considering the real part of complex contributions. Additionally, we swap the voltage boundaries in Equation (11), introducing a canceling sign change in both the numerator and the denominator. This reformulation was chosen to align the presented formula with the implementation in the toolbox.